PreAcher: Secure and Practical Password Pre-Authentication by Content Delivery Networks

Shihan Lin[1], Suting Chen[2], Yunming Xiao[3]*, Yanqi Gu[4], Aleksandar Kuzmanovic[2], Xiaowei Yang[1]
[1] *Duke University,* [2] *Northwestern University,* [3] *University of Michigan,* [4] *University of California, Irvine*

Abstract

In today's Internet, websites widely rely on password authentication for user logins. However, the intensive computation required for password authentication exposes web servers to Application-layer DoS (ADoS) attacks that exploit the login interfaces. Existing solutions fail to simultaneously prevent such ADoS attacks, preserve password secrecy, and maintain good usability. In this paper, we present PreAcher, a system architecture that incorporates third-party Content Delivery Networks (CDNs) into the password authentication process and offloads the authentication workload to CDNs without divulging the passwords to them. At the core of PreAcher is a novel three-party authentication protocol that combines Oblivious Pseudorandom Function (OPRF) and Locality-Sensitive Hashing (LSH). This protocol allows CDNs to pre-authenticate users and thus filter out ADoS traffic without compromising password security. Our evaluations demonstrate that PreAcher significantly enhances the resilience of web servers against both ADoS attacks and preserves password security while introducing acceptable overheads. Notably, PreAcher can be deployed immediately by websites alone today, without modifications to client software or CDN infrastructure. We release the source code[1] of PreAcher to facilitate its deployment and future research.

1 Introduction

Websites have been using web authentication to identify users and protect user accounts for decades. Despite alternatives exist [57, 74, 87], websites still heavily rely on password authentication because of its high usability [28, 33]. According to the estimate by prior studies [28, 101] in 2023, more than one-third of websites contain password login interfaces. However, researchers have expressed the concerns about password authentication's security defects, and a long-standing one is Application-layer Denial of Service (ADoS) attacks [2, 41].

ADoS refers to the attacks where attackers aim to deplete a server's resources by sending merely a small number of requests. Differing from Distributed Denial of Service (DDoS) attacks, the traffic volume in ADoS is much less than that in DDoS, so it will not trigger the commonly used DDoS defense [7, 15]. Password authentication is vulnerable to ADoS attacks because it demands intensive computation. Presently, a server stores a password' hash value instead of the cleartext in its database to prevent password cracking in case the database is compromised [14, 31, 36]. Thus, the server computes the hash value of a password and check it against the record in its database. However, a server must use a slow hash function, such as scrypt [81], PBKDF2 [77], or Argon2 [32], for password hashing, in order to slow down brute force guessing when an attacker captures the database. Therefore, this practice makes password authentication compute-intensive. In § 2.1, we present a proof of concept for such ADoS attacks on the Internet, where merely 150 login requests per second can deplete all four modern CPU cores [10] on a server.

In today's Internet, ADoS can occur during credential stuffing [18, 19] incidents. In such cases, attackers use automated bots to repeatedly attempt login with various combinations of usernames and passwords, aiming to impersonate legitimate users. Akamai reports that it detects an average of 280 million suspicious bot logins per day [18]. Such a large number of illegal logins not only poses security concerns on the safety of user accounts, but also imposes a prohibitively high workload on a web server , due to the slow hashing requirement of password authentication.

Presently, there is no effective defense against the ADoS attacks that exploit password authentication. Existing approaches fail to simultaneously achieve the goal of avoiding server overload, preserving good usability, and providing password secrecy. In practice, websites usually adopt rate limit, CAPTCHA, and two-factor authentication (2FA) in login interfaces [25]. However, attackers can bypass rate limits by switching to different usernames and IP addresses and thus keep sending login traffic. As for CAPTCHA and 2FA, they are criticized for their usability by both users and researchers [39, 47, 48, 55, 64, 99]. Although 2FA prevents user account cracking, it does not reduce the workload of failed

*Yunming Xiao is the corresponding author.
[1] https://github.com/SHiftLin/NSDI2025-PreAcher

logins because the server needs to verify the password as the first factor. Currently, some Content Delivery Network (CDN) providers such as Akamai and Cloudflare have deployed bot detection mechanisms to help websites filter out the login traffic generated from bots [18, 25]. However, these bot detection mechanisms use machine learning and behavior analysis to detect bot requests and cannot deterministically avoid false positive or false negative results. Besides, websites have to allow a CDN to inspect their traffic from users, leading to password exposure to the CDN [35, 101].

In addition, a website may delegate password authentication to a third party by using the service of an authentication provider like Auth0 [3] or using the Single Sign-On (SSO) scheme such as OpenID [21] and OAuth [53]. Such delegated authentication releases the burden of user verification on the server. However, it is not suitable for security-sensitive websites such as banks that cannot trust a third party for user authentication. Additionally, delegated authentication introduces a single point of failure where a vulnerability in a single delegate affects millions of user accounts [89], as exemplified by the recent Google OAuth incident [26].

Motivated by the concerns above, we propose a novel web authentication architecture named *PreAcher* to securely offload the password authentication to a third-party CDN without divulging user passwords to the CDN. Our insight is that traditional password authentication were designed between two parties, *i.e.* a client and a server, before CDNs became an essential component in the web ecosystem. Consequently, they were not able to prevent ADoS attacks and password leakage simultaneously. Differently, we design the PreAcher architecture to incorporate a CDN into the password authentication process. At a high level, PreAcher deploys password-agnostic "*pre-authentication*" on a CDN to intercept failed login requests, relieving the server from unnecessary authentication workload. Specifically, when a user sends a login request to a website, the CDN first pre-filters the password without knowing the exact user password. Only when the password is highly likely to be correct, then will the CDN forward the password to the origin server for final authentication. This design enables a CDN to filter out most of the login requests with incorrect passwords, including those malicious ones intended for ADoS/DDoS attacks and account cracking.

To meet our goal of preventing ADoS attacks while protecting password secrecy, we face two main design challenges. The first challenge arises from enabling a CDN to efficiently pre-filter a password without exposing the actual password to it. In this paper, we assume a CDN as a honest-but-curious adversary where it will operate the functionalities and protocols faithfully but may have unintentional bugs exploited by attackers to peek at or guess user passwords through the messages in transmission [49, 75, 76]. To defend against such an adversary, we combine Oblivious Pseudorandom Function (OPRF) [37] and Locality-Sensitive Hashing (LSH) [58] to design a novel three-party authentication protocol involving a client, a CDN, and a server in PreAcher. This protocol not only hides the password in the login requests but also prevents a CDN from guessing the passwords, thereby efficiently safeguarding and verifying user passwords.

The second challenge concerns compatibility issues. The proposed pre-authentication protocol involves new operations of the CDN, the client, and the web server, which inevitably raises the concerns on compatibility. We elaborately design PreAcher's architecture to be compatible with the current web ecosystem. Specifically, we implement PreAcher's CDN operations on the existing serverless computing services [61], such as Akamai EdgeWorkers [1], AWS Lambda [9], and Cloudflare Worker [8]. These services allow web developers to run customized request processing functions on a CDN's edge servers. Besides, we implement all client operations as a JavaScripts library, which are imported into the web pages and seamlessly installed in the browser when a user visits the website. Therefore, a website can immediately deploy PreAcher unilaterally to safeguard its server.

To sum up, this paper makes the following contributions:

1. We design and implement the first secure and practical three-party password authentication architecture that incorporates a client, a CDN, and a server, *PreAcher*. PreAcher not only protects websites from ADoS/DDoS attacks exploiting the login interfaces but also prevents password exposure to third-party CDNs.
2. PreAcher can be immediately deployed in the current web ecosystem by a website unilaterally without any modification of web clients, CDNs, or hardware.
3. We conduct comprehensive analysis and evaluations on PreAcher, and the results show that it enhances the password security and servers' resistance to ADoS attacks. Besides, the security of the proposed authentication protocol in PreAcher is formally proved (Appendix § D). We also show that it introduces acceptable overhead to the throughput and latency of the login process.

Ethical concerns: This work does not raise any ethical concerns. We use our own server as the victim in all experiments, and the network traffic volume we send to the Internet is small ($< 10Mbps$).

2 Background, Threat Model, and Goals

In this section, we first review the ADoS attacks exploiting password login interfaces. We also present a proof-of-concept experiment of initiating such ADoS attacks towards a web server behind a commercial CDN's protection service. Then we introduce the service model of the CDN used in this paper before we present our threat model and assumptions. Finally, we propose the design goals of our solution.

2.1 ADoS Attacks on the Password Login

Password logins are one of the most popular features on websites. To securely verify user passwords, web servers must compute the hash values of passwords [28], as they store only

the salted hash values [14], rather than the plaintext, due to the risk of data breach [24]. Furthermore, the hash function used in password verification must be slow hashing, such as scrypt [81], PBKDF2 [77] and Argon2 [32], so that an attacker who captures a password database still needs a long time to brute-forcedly crack the passwords [14, 31, 36]. Therefore, password hashing is deliberately designed to require intensive computation. As the modern processors become more powerful, password hashing also evolves to remain workload intensive on those processors. For example, PBKDF2's iteration number is recommended to increase from 1000 in 2000 [77] to 10000 in 2017 [51]. Additionally, new algorithms like Argon2 [32] are computationally intensive on both modern CPUs and GPUs. However, such intensive computation also affects the server's throughput in verifying user login requests, leading to not only possible server overload during concurrent user logins but also the vulnerability of ADoS attacks [2, 22, 23].

Different from DDoS attacks where attackers generate volumetric network traffic to the victim server, ADoS usually needs much less traffic to crash the server because of the asymmetric computation between a client and the server, such as the login request processing as mentioned above. An attacker can send a few login requests to consume up all the server's computing resources and block valid users from login.

Proof of concept. We present an experiment to show the feasibility of ADoS attacks on a server behind a commercial CDN through the login interface. We set up a Virtual Machine (VM) of with 4 vCPUs (Intel Xeon E-2288G [10]) and 16 GiB memory on Azure as the web server. We also employ one of the largest commercial CDNs and pay for its DDoS protection, bot detection, and Web Application Firewall (WAF). The CDN applies a rate limit of around 10 req/sec per IP address on the requests. We use a desktop computer in our university as the client. The server hosts a typical password login page, and every login request is enforced to traverse through the CDN to reach the server. Our goal is to use the client as an attacker to deplete the server's CPUs by login requests.

We implement two password hashing algorithms on the server: PBKDF2 [77] and Argon2 [32]. The former is a traditional algorithm standardized in 2000, while the latter is the state-of-the-art one proposed in 2016. We configure the iteration number of PBKDF2 as 10000, which is the default value of OpenSSL [13] and is recommended by NIST [51]. As for Argon2, we use the recommended parameters by OWASP [14]: 19 MiB of memory, 2 iterations, and 1 degree of parallelism. In both cases of server implementation, the client uses Puppeteer [16] to control Chrome to send 150 login requests per second as the attack traffic to the server. This attack lasts for one hour and the requests contain randomly generated login credentials. The client also adopts the distributed proxies of Bright Data [5] to send requests with different source IP addresses.

In both cases of PBKDF2 and Argon2, we observe that the CDN does not intercept any request, and the server's CPU utilization stays around 100% during the one-hour attack. Besides, the server cannot respond to any other user requests. Therefore, we successfully bypass the commercial CDN's protection and launch ADoS attacks on the victim server.

2.2 CDN Service Model

PreAcher leverages third-party CDNs, such as Akamai and Cloudflare, for pre-authentication due to their extensive networks of distributed edge servers around the world. These edge servers provide sufficient computing and network resources to defend against DoS attacks. However, researchers have raised security concerns about CDNs: to enable a third-party CDN to serve the HTTPS connections from clients, websites have to share their TLS private keys to the CDN [35, 68]. This practice allows a third-party CDN to access the private data transferred in HTTPS connections between users and websites [54, 69], such as user passwords [101]. In addition, researchers also showed that attackers can exploit a CDN's configuration error to fetch the transferred private data [49, 75, 76]. Therefore, in PreAcher's design, we aim for not only defending against ADoS attacks but also preventing the password exposure to CDNs.

In this paper, we refer to the web server hosted by a website and hidden behind a CDN as the "*origin server*" or simply "*server*". We use the term "*user*" to denote a website's user rather than a CDN's user. A user accesses a website through a web "*client*" such as a browser. We describe an attacker as being "*inside*" a CDN when it exploits a vulnerability within the CDN to gain internal privileges, or when it is a rogue insider of the CDN provider. For simplicity, we sometimes use "*CDN*" to refer to an attacker inside a CDN. Additionally, an attacker "*outside*" of a CDN refers to any malicious entity except for the CDN itself.

2.3 Threat Model and Trust Assumptions

Attackers inside CDNs. Prior studies [54, 69, 101] describe two threat models of a CDN as below:

1. **Passive attacker**: A CDN will honestly provide the functionalities it promises, but its components such as the log and the storage may contain unintentional bugs like configuration errors. An attacker can exploit these bugs to access to the data, but cannot modify a CDN's behavior. The attacker can eavesdrop on the messages transferred by a CDN but *cannot* tamper with, duplicate, or fabricate any message. For example, the attacker can observe the passwords or attempt to infer the passwords from the transferred messages. This model is also called a "*honest-but-curious*" model.
2. **Active attacker**: A CDN's component such as the control plane contains severe vulnerabilities that an attacker can exploit to not only eavesdrop on transferred messages but also tamper with, duplicate, and fabricate the messages. An active attacker is more powerful than a passive attacker.

In PreAcher, we assume a *passive attacker* inside a CDN due to the following reasons. 1) Researchers have shown that passive attackers could exist in commercial CDNs, such as web cache deception attacks [49, 75, 76], where attackers deceive CDNs to accidentally cache some users' private data and grant public access to the data. 2) Prior studies have proposed methods to ensure the integrity of CDN-hosted content based on a secure DNS channel [46, 69], which prevents a CDN from tampering with the message without being detected. PreAcher can adopt these methods to further defend against an active attacker. We leave the defense against active attackers for future work.

We note that a CDN will not launch DDoS/ADoS attacks on the hosted websites in our study, as the CDN is assumed to be a passive attacker.

Attackers outside CDNs. An attacker outside a CDN can actively manipulate the messages it interacts with. For example, it may gain access to a client's Wi-Fi router and alter, duplicate, or forge the transferred messages. We also consider it has the capability to launch ADoS or DDoS attacks. This may involve sending send a large number of login requests with various combinations of usernames and guessed passwords to the login interface of a website.

Origin servers. Since the origin server is the source of all web content, we trust the server for all users' sensitive data. We do not consider data breaches or web providers' misoperations on the server in this paper.

ADoS-resist registration. We assume the server's registration interface is resistant to ADoS. This assumption is achievable in reality. Firstly, since the registration is a one-time cost, a website can adopt complex CAPTCHA on the registration page without affecting the user experience of future visits. Besides, for those web services that are highly security conscious, the web provider can require a user to register an account offline. For example, a user may need to visit a local branch to open a bank account.

Trusted Computing Base. We trust the implementation of existing cryptographic libraries in the programming languages. We also trust the implementation of the hardware, browsers, and operating systems used by the client and server.

2.4 Design Goals

In this paper, we aim to design and implement a password pre-authentication system on CDNs with the following goals:

1. **Security**: Our system focuses on securing password authentication. It should prevent the ADoS attacks exploiting the login interface, while avoiding password exposure to CDNs or other attackers. Besides, the system should retain the CDN's security benefits such as DDoS protection and WAF. However, we do not aim to prevent ADoS attacks using other channels beyond password authentication, nor do we protect other sensitive data or private communications handled by the current CDNs beyond passwords.
2. **Compatibility**: Our system should be compatible with the current web ecosystem. Its deployment should not involve multiple stakeholders on the Internet. Our design achieves this by enabling websites to deploy the system unilaterally. The system does not require any modification on current browsers, CDN infrastructure, or operating systems. It only requires updating the website's login page and processing.
3. **Efficiency**: Our system should not introduce much overhead to the login procedure from the perspectives of throughput and latency.

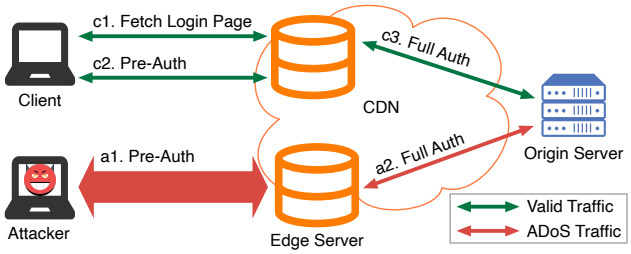


Figure 1: **PreAcher architecture. The width of arrows shows the traffic volume.**

Achieving these three goals simultaneously is non-trivial. We utilize both techniques of authentication protocols and the features of current web development practice and thus finally design and implement PreAcher shown in this paper.

3 Design Rationales

In this section, we first illustrate an overview of PreAcher. Then we discuss the design rationales of PreAcher.

3.1 Overview

The core of PreAcher is an authentication protocol involving three parties: the client, CDN, and the server. It includes a *pre-authentication* stage on the CDN and a *full authentication* stage on the server, which we will elaborate in § 4.

Figure 1 shows an overview of PreAcher. All components of PreAcher run on the application layer of the Internet for compatibility. When a user visits a website and intends to login, the web client (browser) fetches the HTML files of the website's login page from the CDN (`c1`). Since we assume a passive attacker in the CDN (§ 2.3), the CDN cannot modify the login page and thus cannot conduct phishing on user passwords. After the user types in her username and password on the web page, the client communicates with CDN to pre-authenticate the user (`c2`), which may include multiple round trips. Only if the client passes the pre-authentication, the CDN will forward the client's request to the origin server for full authentication (`c3`). If the server also successfully verifies the user's identity, it will return an indication of success to the CDN and the CDN finally notifies the user of a successful login. Otherwise, the client is rejected to login.

When an attacker attempts to compromise some accounts, it can instrument programs to directly send usernames and passwords to the CDN and start the pre-authentication (`a1`), without loading the HTML files into a browser. In most cases,

the pre-authentication will fail, and the CDN immediately rejects the requests without forwarding them to the server. Therefore, if the attacker sends login requests to initiate ADoS on the server, the CDN will filter out most of them to protect the server. Since the CDN conducts a pre-authentication which does not ensure 100% correctness of the credentials, some attack requests may still be forwarded to the server for full authentication (a2). However, the server will reject this portion of requests. Besides, we elaborately design the authentication protocol to ensure that the amount of these requests is small enough and will not lead to ADoS on the server.

3.2 HTTPS Compatibility

PreAcher is compatible with the current web ecosystem, and thus all communications among the client, the CDN, and the origin server use HTTPS connections, including the requests/responses of page loading, pre-authentication, full authentication, *etc.*. We keep current TLS private key-sharing practice between CDNs and websites, so the CDN can still access to the HTTPS payloads. In contrast, an attacker outside of a CDN cannot break HTTPS and view the content, since it does not have the TLS private key.

3.3 Private Communication After Login

PreAcher does not aim to protect general private communication between the client and the server. Instead, PreAcher focuses on preventing the password exposure and ADoS/DDoS attacks caused by current password authentication. A website can adopt the existing proposals, InviCloak [69], to build a secure channel between the client and server with the existence of a CDN and thus transfer the private data securely through the channel. Besides, after a success login, websites usually grant the client authentication cookies to avoid logins for future visits. PreAcher does not provide the protection of these cookies either, but websites can also use InviCloak to protect the authentication cookies.

3.4 Password vs Additional Key

Some design ideas may require a client to hold an additional key—such as a private key—after the registration. We decided not to adopt these designs because they conflict with current user behaviors: users are required to remember or save additional materials beyond the passwords. Although a web client can help a user to store the additional key, a user may change her web client. Thus, PreAcher adopts a design compatible with the current user behavior of memorizing the username and password only, which is arguably the most universal authentication method.

3.5 Preventing Offline Dictionary Attack

In an offline dictionary attack [29, 34, 82], an attacker obtains a dictionary of user passwords, enumerates all password candidates in the dictionary, and verifies the guessed password offline (locally) without communications with the server. Such an attack requires the attacker to find an approach to determine the correctness of the guessed password locally. For example, an attacker may obtain the hash value and the salt of the actual password so that it can compare the hash value of the guessed password with the actual hash value to know the correctness. Therefore, the attacker can enumerate all passwords in the dictionary locally until it finds the correct one, and it will not be detected by the server.

The defense against such an attack becomes particularly challenging when we consider a passive attacker inside a CDN in PreAcher's design. Generally, if a CDN can fully authenticate users to filter out all invalid requests, then the CDN can launch offline dictionary attacks. This is because the passive attacker can observe all transmitted messages and all computational states of the CDN during a legitimate user login. With this information, for each password the attacker guesses, it can simulate the authentication protocol between the client and the CDN to determine the password's correctness. Note that such an attack remains within the bounds of passive attacker behaviors since the simulation relies solely on observed data and is conducted offline.

Furthermore, if we prevent the CDN from obtaining any information about password correctness to thwart such offline dictionary attacks, the CDN will be unable to filter out the failed login requests for the server.

We addresses such a challenge based on two observations:

1. Although offline dictionary attacks are hazardous, *online* password-guessing attacks are considered much less risky since they are detectable and can be easily rate-limited by the origin server.
2. Currently, the state-of-art dictionary generation algorithms generate the passwords based on a user's historical passwords [80,102] or a user's personal information [91]. Thus, the password candidates in a dictionary should be similar to each other to some extent [56].

Therefore, in PreAcher, instead of granting a CDN the capability of determining the password correctness, we propose to only enable the CDN to determine the similarity between the provided password and the actual password. The CDN can still help the server filter out massive failed login requests with erroneous passwords that are not similar to the actual password, and it forwards the login requests with probably correct passwords to the server for further authentication.

Such a design prevents offline dictionary attacks initiated by a CDN because a CDN cannot verify similar passwords offline. By PreAcher's design, for those similar passwords in the dictionary, CDN cannot determine which one is the correct one, and it has to query the server to verify every similar password. Therefore, CDN can only guess the password online with the server.

The design still prevents ADoS attacks. For attackers outside the CDN, if it randomly generates the passwords, the portion of passwords that can pass the CDN's pre-authentication will be very small (see § 5). Moreover, as we analyze in

§ 6, it is difficult for attackers outside the CDN to identify the passwords that can reach the origin server and trigger an ADoS attack. Furthermore, even if they gain access to such passwords, since the password authentication is enforced to be online and shares a pattern of similarity, the attack is detectable by the server.

We describe the algorithm to enable a CDN to determine the similarity of passwords in § 5, which is based on Locality-Sensitive Hashing (LSH).

4 Authentication Protocol

As previously discussed, our protocol consists of two stages: the pre-authentication on the CDN and the full authentication on the server. Although the idea of this two-stage protocol is simple, instantiating such a protocol is non-trivial. In this section, we first present the protocol used in PreAcher, and then we discuss an intuitive but less efficient design.

4.1 PreAcher Protocol

The core idea of our protocol is to combine Oblivious Pseudo-random Function (OPRF) [37] and Locality-Sensitive Hashing (LSH) [58] to enable pre-authentication at a CDN while limiting its knowledge of password correctness.

OPRF is a protocol that allows two parties (Alice and Bob) to obliviously evaluate a PRF function $F(K;X)$, where Alice inputs K and learns nothing, and Bob inputs X and learns $F(K;X)$ without knowing K. In this paper, we use the 2HashDH OPRF proposed in prior work [59]. OPRF is also used by existing Password Authenticated Key Exchange (PAKE) protocols like OPAQUE [60]. Our protocol is inspired by OPAQUE but is different from its design in two aspects:

1. OPAQUE is designed for two-party authentication between the client and server. We extend OPAQUE to three parties.
2. By adding the CDN into the protocol, OPRF itself cannot solve the offline dictionary attacks by the CDN. Our protocol incorporates LSH to defend against the attacks.

Furthermore, the security of this proposed new protocol is also formally proved. We describe the workflow of the protocol in this section, and present the proof in Appendix D.

We first describe the notations used in the protocol below before we elaborate on the protocol details.

- The client obtains the username u and the password p from a user's input. The server generates a new key pair $\langle \mathsf{pk}_s, \mathsf{sk}_s \rangle$ during the deployment of the system. Note that this key pair is different from the TLS key pair.
- For clarity, the symbols with a subscript "$_u$" are related to a specific user u, and the symbols with a superscript "$'$" are related to pre-authentication, *e.g.* a user's key pair used for pre-authentication $\langle \mathsf{pk}'_u, \mathsf{sk}'_u \rangle$.
- The protocol builds upon a Diffie-Hellman cyclic group $G = \langle g \rangle$, where g is the generator, and n is the bit length of the group.
- H_1 is a cryptographic hash function, while H_2 is password hashing. *LSH* denotes a locality-sensitive hashing.

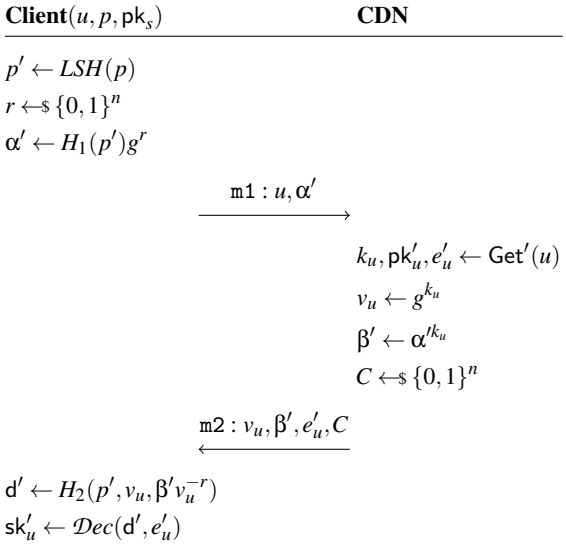


Figure 2: **First round of PreAcher login.**

- The protocol uses a CPA-secure symmetric encryption scheme $(\mathcal{K}g, \mathcal{E}nc, \mathcal{D}ec)$ and a CCA-secure asymmetric encryption scheme $(\mathsf{Kg}, \mathsf{Enc}, \mathsf{Dec})$.
- We use a standard digital signature scheme $(\mathsf{KGen}, \mathsf{Sign}, \mathsf{Vf})$ with existentially unforgeability under chosen-message attack (EU-CMA).
- $\mathsf{Save}(u, \cdot)$ (resp. $\mathsf{Save}'(u, \cdot)$) denotes saving the input into a user-associated record stored by the origin server (resp. CDN), while $\mathsf{Get}(u)$ and $\mathsf{Get}'(u)$ denote the retrieval of the corresponding records associated with a user u.

The protocol is divided into two phases: registration and login. During the registration, we assume trust on first use so the client can communicate directly to the server. The server first generates a random number k_u. Then the client and the server will use OPRF to compute a symmetric key d' through the password and k_u. The client generates a key pair $\langle \mathsf{pk}'_u, \mathsf{sk}'_u \rangle$, and encrypts the sk'_u into an envelope e'_u with the key d'. The server sends $k_u, \mathsf{pk}'_u, e'_u$ to the CDN after the registration, and the CDN will use them to help the server pre-authenticate users and filter out invalid login requests. The client also registers the user with the server by traditional password hashing (without exposing the password to the CDN), which will be used for full authentication on the server during login. Given space constraints, we leave more details of registration in Appendix A. For the rest of this section, we focus on the login protocol.

Initialization. A user inputs u and p to the client. The client obtains pk_s, and the server holds the corresponding sk_s. The server can delivered pk_s to the client with integrity by embedding it in the HTML of the login page as discussed in § 3.1. Besides, the CDN already received corresponding $k_u, \mathsf{pk}'_u, e'_u$ for registered users from the server.

Generating the secret key from the password. Similar to the registration, the client obtains the secret key d' through

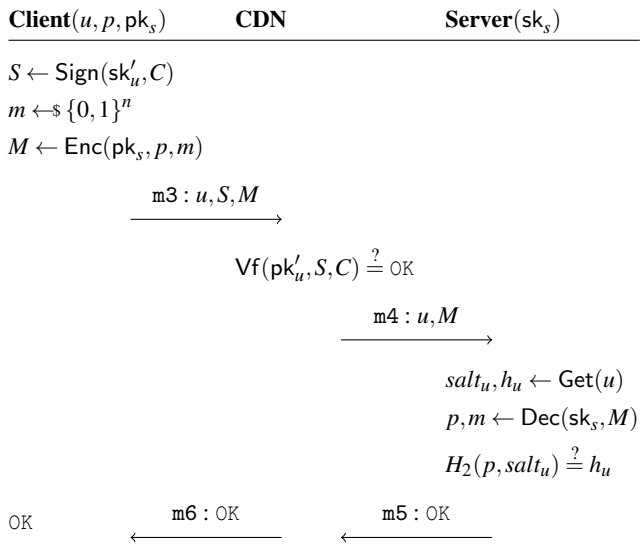


Figure 3: **Second round of PreAcher login.**

the combination of OPRF and LSH, but it communicates with the CDN instead of the server as shown in Figure 2.

Specifically, the client uses LSH to map the password p to a pseudo-password p'. Then it generates a random nonce r and computes α'. After receiving u and α' from the client (m1), the CDN retrieves k_u, pk'_u, and e'_u from the storage according to u. Then it computes v_u and β', and returns them to the client with e'_u and a challenge C (m2). The client computes d' through the same function in the registration.

Retrieving the private key from the envelope. With d', the client can decrypt e'_u to obtain the private key sk'_u which is generated in the registration. This completed the first round of PreAcher login.

Pre-authentication and full authentication. In the second round, the client computes the credentials for pre-authentication and full authentication, respectively. Then it sends them to the CDN and the server together to save the RTTs of the protocol. The procedure is shown in Figure 3.

For pre-authentication, the client signs C with sk'_u and send the signature S to the CDN (m3). Meanwhile, for full authentication, it encrypts p and a nonce m with pk_s and sends the encrypted message M to the CDN without concerns of exposing p (m3). Since the CDN stores pk'_u, it can verify the signature S. The pre-authentication succeeds if and only if the verification succeeds. If the pre-authentication fails, the CDN stops the protocol and sends an error message to the client, which is not shown in the diagram. When the pre-authentication succeeds, the CDN sends u, M to the origin server for full authentication (m4). Note the CDN cannot decrypt M to know p since it does not know sk_s.

When the server receives requests for full authentication, it retrieves $salt_u$ and h_u from records and decrypts M to authenticate the password p by comparing the calculated hash value with h_u. The authentication result is returned to the CDN (m5) and then to the client (m6).

Summary. The protocol uses two different asymmetric key pairs for pre-authentication and full authentication, respectively. For pre-authentication, the client's private key is enclosed in an envelope stored on the CDN with a symmetric encryption key. The encryption key is computed by OPRF from the password in the login. We combine LSH with the OPRF to limit the CDN's capability of guessing passwords. We will discuss how LSH achieves this in § 4.2.

As shown in the protocol, the pre-authentication uses the signature to verify the user, while the full authentication uses the traditional password hashing where the server sees the password and checks against the stored hash value. We expose the password to the server because we trust the server in our threat model (§ 2.3). Besides, we keep such a traditional method in the server to be compatible with the popular authentication implementation in practice. In addition, if we use signature verification and hide passwords from the server, one more RTT will be added to the protocol since a challenge number should be delivered to the client after the pre-authentication. Nevertheless, web operators can easily modify this full authentication approach if they consider the traditional method does not fit their demands.

4.2 Intuitive but Less Efficient Design

In the early stage of our study, we adopted straightforward hashing methods for pre-authentication. Overall, the client hashes p' to obtain $H_1(p')$. The CDN computes password hashing of $H_1(p')$, *i.e.* $H_2(H_1(p'))$, and checks it against with the one in the storage. We call such a double-hash protocol as "*DuoHash*". We illustrate the detail protocol of DuoHash in Appendix C.

However, we found that DuoHash is less efficient in practice, since the protocol runs a slow password hashing on the CDN for pre-authentication. Besides, this slow hashing cannot be replaced by a usual hash function like SHA-256, because such a replacement accelerates the brute-force guessing of p' when an attacker captures the CDN's storage [14,31,36]. For example, using OpenSSL implementation on our testbed, SHA-256 is over 50,000 times faster than PBKDF2. With p', the attacker can simply construct the login requests that pass pre-authentication and thus launch ADoS attacks on the server. Therefore, password hashing is necessary in the double-hash design, leading to a significant performance downgrade of the CDN's edge servers. As a defense of ADoS and DDoS for the origin server, the CDN's operations should be as efficient as possible, so we opt for the OPRF design in this paper. Web developers could adopt DuoHash if they find it suitable for their applicatons, as outlined in our comparison of DuoHash and PreAcher in § 8.

5 Locality-Sensitive Hashing

As presented in § 4.1, we use LSH to map the password p to a pseudo-password p', and use p' in the pre-authentication. The

rationale behind this design is to reduce the CDN's ability to determine password correctness, thereby preventing offline dictionary attacks from the CDN as discussed in 3.5.

The LSH is designed to map a set of similar passwords to the same p' value, and thus, a portion of erroneous passwords can also pass the pre-authentication. Specifically, an attacker inside the CDN may launch offline dictionary attacks by generating a dictionary of password guesses and testing them locally to see which ones pass the pre-authentication. However, by using LSH, the best this attacker can do is to find that a portion (determined by LSH) of passwords are hashed to the correct p', and they are all possibly the actual password. Thus the attacker has to enumerate this portion of passwords and send online requests to the server for full authentication, but the server will reject this login request unless the correct password is guessed. Leveraging LSH, the CDN only knows that a provided password may be similar to the correct one locally. This achieves our purpose of enforcing the CDN to conduct online queries to know the correctness.

For the passwords in a dictionary, we expect to see many of them mapped to the p' of the actual password, so that CDN gains little information through enumerating the passwords in the dictionary. However, to defend against ADoS attacks, we expect to reduce the probability of the passwords in the attack traffic mapped to the actual p' so that the CDN can filter out most failed logins. These two requirements align well with the design of LSH, and we can tune the parameters of LSH to balance the CDN's ability of inference and authentication.

In PreAcher, we use weighted K-mer MinHash [72] as the algorithm of LSH. A string s of length N is first converted to lowercase and then split into $N-K+1$ substrings called "K-mers" by a sliding window of length K. These K-mers form a sequence S, following the order of their positions in s. Formally, let $s[i:j]$ denotes the substring of s from index i to index j. We have

$$S = (s[1:K], s[2:K+1] \ldots s[N-K+1:N]) \quad (1)$$

Then we use weighted MinHash [96] to generate the hash value from S as follows. Since a K-mer (denoted as m) may appear multiple times in S, we use $m^{(w)}$ to denote w-th appearance of m in S. We use w as the weight in weighted MinHash. Then we adopt HMAC [63] with SHA-256 to hash the K-mers in S. Thus, we have a set T consisting of the hashed K-mers.

$$T = \{\mathsf{HMAC}(u, w \parallel m) \mid m^{(w)} \in S\} \quad (2)$$

In Formula 2, we use $\parallel$ to denote concatenation. Finally, we take the minimum element in T as the hash value of LSH.

We emphasize that our LSH design is preimage-resistant. Specifically, an attacker only knows that there exists a cluster of similar p to be mapped onto the same p', but it is difficult to determine the exact p that corresponds to a specific p'. This property originates from our two designs:

1. We use HMAC with SHA-256 to construct the LSH as shown above. As SHA-256 is preimage-resistant, the LSH is also preimage-resistant.
2. We include the username u in HMAC's input. Thus, the mappings from the p to p' vary across different users, further enhancing the resistance to preimage attacks.

We then analyze the collision probability of this LSH. Let c be the alphabet size of a password. Since the K-mer has a length of K and the hash value is generated from K-mers, the size of the hash value space is c^K. Therefore, in the case of ADoS/DDoS initiated by an attacker outside of a CDN, it generates passwords randomly. Then the probability of the collision between a generated password and the actual password is $P_{col} = \frac{1}{c^K}$. This is the fraction of the attack requests that pass the pre-authentication by the CDN. For example, let $K = 4$ and $c = 66$, where c includes 26 case-insensitive letters, 10 digits, and 30 special characters allowed by Linux passwords, then P_{col} is less than 10^{-7}. It means that PreAcher can filter out almost all DoS traffic.

To identify the correct password, an attack inside the CDN could offline determine the password candidates that are mapped to the actual p' and then send these candidates to the server for online verification. In this case, the server will see multiple failed login requests of an account from the CDN. For an account, the server should consider it is under attacks if more than Q times of failed logins happen on the server. The server could temporarily freeze the user account and ask the CDN to inspect the possible insider attackers. However, these frequent failed logins may also come from a valid user's erroneous input. Thus, the server can tune Q to adjust the sensitivity to the attacks. We run an empirical experiment simulating this scenario to show the defense effectiveness of LSH as below.

We collect 5000 distinct users and their passwords from 4iQ dataset [38]. For each user, we collect two passwords, one is regarded as a historical password leaked to a CDN, the other one is regarded as the actual password of the user's account on a website served by the CDN. Suppose the website adopts the CDN for pre-authentication, and the CDN aims at cracking those 5000 user accounts in this website through the collected historical passwords. Suppose the CDN adopts a state-of-the-art dictionary generation algorithm, *pass2path* [80] for password guessing. For each account, it uses this algorithm to generate a dictionary of 10,000 passwords from the historical password. The CDN can locally filter the password candidates that pass pre-authentication and send them to the server for full authentication. When the pre-authentication does not adopt LSH or the website opts for full authentication on the CDN, as long as the dictionary contains the correct password, the CDN will find it locally and crack the account successfully. However, with LSH, only if less than Q password candidates are left after the offline filtering can the CDN ensure itself cracks the account through less than Q online queries without being detected by the server. We compute the success rate of such undetected cracking on 5000 accounts for different values of Q and K in our LSH.

Figure 4 shows the success rate of cracking. Firstly, the

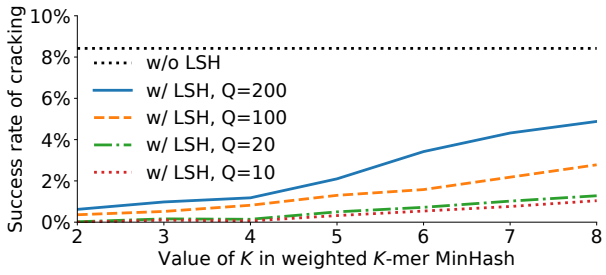


Figure 4: **The CDN's success rate of cracking accounts varying with K and Q in LSH. When LSH is not used (the black line), the success rate does not depend on K or Q.**

black line is about 8.42%, indicating a relatively high chance that the CDN can find out an account's password offline when LSH is not adopted. However, the success rate is dramatically reduced with LSH applied. It is less than 1% When $K \leq 4$ and $Q \leq 100$. The success rate also rises as K and Q increases, because larger K leads to fewer password collisions with p' in LSH, and larger Q allows more trials from the CDN. Nevertheless, a small K also reduces the effectiveness of the ADoS/DDoS defense of the CDN, and a small Q also downgrades the tolerance of the typos from valid users. Therefore, a server adopting PreAcher should select K and Q properly to defend against both DoS attacks and password cracking and reduce the false alarms. According to Figure 4, we set K as 4 and Q as 20 by default in PreAcher, since the cracking rate is less than 0.20% and almost all DoS traffic cannot pass the pre-authentication ($P_{col} < 10^{-7}$). Websites deploying PreAcher can adjust K and Q based on their demands.

We note that the 0.20% cracking rate is not introduced by PreAcher but results from users frequently reusing similar passwords across websites. A website without PreAcher faces the same cracking rate when an attacker outside of the CDN conducts online password guessing. Additionally, without PreAcher, the CDN sees the passwords directly. With OPRF but not LSH in PreAcher, the CDN can guess passwords offline with a success rate of 8.42% (Figure 4). Our LSH design contributes to the reduction from 8.42% to 0.20%.

6 Security Analysis

In this section, we analyze the security properties of PreAcher and explain how our design defends against attacks. We have formal security proof for our protocol, and we provide a proof sketch in Appendix D.

Registration security: As discussed in our threat model 2.3, we consider registration is ADoS-resist. Besides, the server still hides behind the CDN for registration, and the CDN can provide the DDoS defense. Moreover, the password is not exposed to the CDN during the registration according to the protocol shown in Appendix A.

User impersonation: If an attacker tries to impersonate a user without knowing the correct password, it will be rejected by either the CDN or the server. However, there exist cases in which the attacker provides a wrong password but obtains the correct p' because LSH maps the wrong and correct passwords to the same p'. In such cases, the attacker can pass the CDN's pre-authentication because he can compute the correct d' to decrypt e'_u. However, he is guaranteed to fail the authentication on the server since the wrong password cannot pass the password check against the stored hash.

Password invisibility to the CDN: With our protocol, the CDN cannot observe user passwords, as the passwords are neither transferred in plaintext nor decryptable by the CDN during the communication. During the login, the oblivious nature of OPRF allows the client to retrieve the stored private key sk'_u. Then the client and the CDN use $\langle sk'_u, pk'_u \rangle$ for pre-authentication. Moreover, the client encrypts the password with pk_s during full authentication. As a result, both authentication stages ensure the secrecy of the password. We provide two examples of attack attempts in Appendix B to demonstrate how this protocol can defend against a passive attacker inside the CDN.

ADoS/DDoS attacks: PreAcher prevents ADoS/DDoS attacks by filtering out most failed login requests by pre-authentication on the CDN. When an attacker generates brute-force login requests to overload the server, the server is never involved in the pre-authentication while the CDN will intercept the requests accurately.

An attacker may try to find passwords that pass pre-authentication but fail the full authentication to repeatedly involve the server and drain its resources. However, the attacker cannot identify such passwords in PreAcher. Both authentication stages return the same failure message. When a wrong password is guessed, the attacker cannot tell if the failure is due to pre-authentication or full authentication. Therefore, attackers outside the CDN cannot find passwords accepted by pre-authentication to launch ADoS attacks. While the CDN knows the pre-authentication result, as assumed in § 2.3, a CDN would be improbably involved in ADoS/DDoS attacks.

An attacker may also register an account and use the account's credentials to launch ADoS. The origin server can defend against such attacks by monitoring the login behavior of users. If a user repeatedly logs into her account in a very short time, the website can block the account.

Dictionary attacks: The key idea of PreAcher's defense against dictionary attacks is to force attackers to guess passwords online, and the server enforces a rate limit on the failed login attempts to detect such attacks.

For an attacker outside of the CDN, since it does not know k_u, it can only conduct online guessing for each of its password guesses. The only information the attacker gains is (1) the OPRF output which is indistinguishable from random bits to the attacker due to the pseudorandomness of OPRF and (2) envelops e'_u which the attacker cannot learn anything from due to the semantic security.

PreAcher also prevents offline dictionary attacks from a passive CDN attacker. As discussed in § 3.5, we have to

allow a CDN to guess p' offline to enable it to conduct pre-authentication. However, even with the knowledge of p', the CDN cannot guess p easily online due to our LSH design. As shown in § 5, our design of LSH reduces the cracking rate from 8.42% to less than 0.20%, even when a CDN uses a user's historical password and a state-of-the-art password generation algorithm to make 20 guesses for a single account.

Credential stuffing: Credential stuffing is a type of dictionary attacks where the attacker collects the existing dataset (dictionary) to guess the password. Since PreAcher defends against dictionary attacks, it also mitigates credential stuffing. In addition, PreAcher provides the benefit of reducing the origin server's workload in credential stuffing since most incorrect credentials are filtered out by the CDN.

Replay attacks: For the attacker outside of the CDN, it cannot replay any message of a valid user, as all communications in PreAcher are inside HTTPS. Moreover, the CDN cannot replay messages either, since it is assumed as a passive attacker (§ 2.3). Furthermore, the server can also record the recent messages of M (Figure 3) to prevent the CDN from replaying them, which is adopted by prior work [69].

Side-channel attacks: An attacker outside of the CDN may attempt to identify the passwords passing the pre-authentication by a side channel so that it can launch ADoS attacks with these passwords.

For example, an attacker may exploit response times by sending login requests with different passwords, as failed pre-authentication attempts are rejected by the CDN and replied faster than the successful ones. However, PreAcher can mitigate such a timing side channel by delaying failed responses, masking the pre-authentication results. Such a delay should approximate to the latency between the CDN's edge server and the origin server. Moreover, valid users remain unaffected, since this delay is not applied to successful logins.

In additional, an attacker could send many login requests with a specific password and infer the the server's workload through the website's page load time. A significant increase in the page load time indicates this password pass the the pre-authentication. PreAcher can generally prevent this side-channel attack as well as others aiming at differentiating pre-authentication results because

1. The values of p' used for pre-authentication spread in a large space ($> 10^7$, see § 5), requiring many trials to guess a correct p'. Moreover, each trial takes considerable cost as it involves one or more online requests.
2. The LSH mappings are different across users, and the server freezes an account after Q failed full authentication attempts. Thus, an attacker is limited to Q requests per p' inferred through the side channel, and it must repeat the trails of guessing p' across numerous users to gather enough p' to carry out a meaningful ADoS attack.

Finally, researchers have discovered other side channels that may compromise user data, such as through CPU cache [107], keystroke video [104], improper app isolation [105]. These attacks pose threats not only to PreAcher but also to many other systems. Addressing these attacks is beyond the scope of this paper and requires broader efforts extending beyond websites and CDNs.

7 Implementation

We provide a prototype implementation of PreAcher to facilitate the deployment and future research.

We implement the client operations in the protocol as a JavaScript library so that web developers can easily apply PreAcher by importing the library into their login web pages. The implementation does not require any modification or extension of existing browsers. Therefore, the websites can be seamlessly upgraded without cooperation from users.

We implement the server in C++ with Sogou Workflow framework [17], which is an efficient C++ server engine used in the industry to serve billions of requests per day. Web developers can adapt our server implementation into their web servers, or they can simply run our implementation as a separate process and redirect the received login requests to the process by setting up a reverse proxy [62, 83].

As for CDN operations, we implement them as a JavaScript code snippet running on a CDN's serverless computing service. Currently, CDN providers offer serverless computing to allow websites to run customized code at their edge server to process requests, such as Akamai EdgeWorkers [1], AWS Lambda [9], and Cloudflare Workers [8]. Therefore, our implementation runs seamlessly without requiring changes to the CDN infrastructure, despite introducing new operations on CDN edge servers. Web developers can deploy our JavaScript code snippet directly to the CDN to enable pre-authentication, as all the mentioned CDNs support JavaScript.

For cryptography operations, we adopt the existing libraries Web Crypto API [94] and libsodium [11] in JavaScript implementation and OpenSSL [12] in C++ implementation.

Overall, we elaborately design and implement PreAcher so that it is compatible with the current web ecosystem and can be immediately deployed by websites. Furthermore, PreAcher supports incremental deployment across websites since each website can unilaterally manage its deployment progress.

8 Evaluation

In this section, we evaluate PreAcher's efficacy in defending against ADoS attacks and the overhead introduced to throughput and latency on both the testbed and the Internet.

8.1 Experiment Setup

We set up a testbed on Azure using three VMs as a client, a CDN, and an origin server. Each VM has 4 vCPUs and 8 GiB of RAM and runs Ubuntu. As for Internet experiments, we deploy PreAcher on a commercial CDN, *i.e.* Cloudflare [6] and evaluate it using Azure VMs as the client and the server.

Besides PreAcher, we implement three strawman methods for comparison. The first one models the current password au-

thentication practice in a CDN-enabled website. The CDN's edge servers forward login requests to the origin server for authentication. Although this method directly exposes passwords to the CDN, it is simple and widespread [101]. We call this method as "*Baseline*" in this section. Another method is "DuaHash" discussed in § 4.2. Theoretically, DuaHash also prevents password exposure and ADoS attacks. Finally, we implement a method based on Intel SGX [40]. Specifically, we adopt Occlum [85] to run a traditional password hashing to authenticate users in SGX. We call this method as "*SGX-CDN*', which theoretically prevents password exposure and ADoS attacks but requires hardware support.

PBKDF2 is used for password hashing as shown in § 2.1.

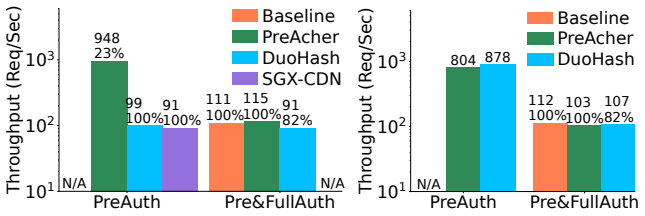
8.2 ADoS Defense

Method	Testbed		Internet	
	w/o ADoS	w/ ADoS	w/o ADoS	w/ ADoS
Baseline	100	0	99	0
PreAcher	100	97	100	100
DuoHash	91	0	98	97
SGX-CDN	90	1	N/A	N/A

Table 1: **The successful logins per second for four methods with and without ADoS attackers. Since Cloudflare does not deploy SGX, we cannot measure the SGX-CDN on Cloudflare.**

We simulate ADoS attacks on the server and compare the efficacy of four proposed methods on the testbed and the Internet. We set up two processes on the client VM to act as valid users and an attacker, respectively. The user process sends login requests at the speed of 100 req/sec to simulate the case where 100 valid users intend to login to the website per second. All requests contain correct usernames and passwords. As for the attacker process, it sends login requests at the speed of 400 req/sec to simulate the case where an attacker initiates ADoS traffic to crash the server. These requests contain erroneous credentials since the attacker does not know the correct one. We run the experiments in two settings on both the testbed and the Internet. In the first setting, only the user process sends requests, representing a normal network condition without attacks (`w/o ADoS`). The second setting contains both the user process and the attacker process, representing a network condition under the ADoS attacks (`w/ ADoS`).

Table 1 shows the throughput (the number of successful logins per second) of each method. On the testbed without attack traffic, the server can respond to the user process timely with any of the methods deployed. However, during ADoS attacks, only PreAcher can maintain high throughput, *i.e.* 97 req/s, while the other methods fail to serve nearly all valid requests. For the baseline, it does not conduct the pre-authentication on the CDN and all attack requests will be forwarded to the server for processing. As for DuoHash and SGX-CDN, the pre-authentication overloads the CDN VM due to the intensive computation for password hashing, and thus the CDN cannot forward the valid requests to the server.



(a) Testbed (b) Internet

Figure 5: **Throughput of pre-authentication (`PreAuth`) and the whole system (`Pre&FullAuth`) for four methods. "N/A" indicates the absence of corresponding operations. Numbers above bars represent the values of throughput and CPU utilization. Some CPU utilizations are missing because Cloudflare does not provide the information.**

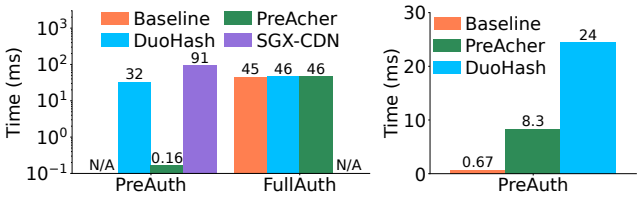
For the experiments on the Internet, the methods also serve user requests normally when there are no attacks. The throughput of DuoHash is improved because Cloudflare has plentiful CPU resources for password hashing. Due to the same reason, DuoHash can serve valid users with 97 req/s even under attacks. PreAcher also prevents ADoS attacks when it is deployed on Cloudflare, and it consumes less CPU than PreAcher as we will presented in § 8.4.

8.3 Throughput

We separate the experiments in this section into two groups: one measures the throughput of failed logins rejected by the CDN, and the other one measures the throughput of successful logins accepted by the server. The former represents the pre-authentication's throughput on the CDN, and we indicate this group as "`PreAuth`". The latter represents the whole system's throughput including both authentications on the CDN and the server, indicated by `Pre&FullAuth`. Specifically, we use `wrk` [20] at the client to send login requests to the CDN by 64 parallel connections (`wrk -c64`).

Figure 5a and Figure 5b show the throughput of the two groups for the four methods on the testbed and the Internet, respectively. Firstly, the baseline does not use the CDN for pre-authentication, and SGX-CDN does not run authentication on the server (it totally relies on the CDN for authentication), so we put "N/A" in the corresponding positions of bars. Figure 5a shows that PreAcher achieves a much higher throughput (948 req/s) than DuoHash (99 req/s) and SGX-CDN (91 req/s) in the `PreAuth` group. Moreover, it takes PreAcher only 23% of CPU to handle these requests, while DuoHash and SGX-CDN use up all CPU resources (100%). These results demonstrate the high efficiency of PreAcher's pre-authentication, which explains the effectiveness of its ADoS protection in § 8.2. The pre-authentication throughput of DuoHash and SGX-CDN is due to the slow password hashing, and SGX-CDN also introduces the overhead of SGX. As for the `Pre&FullAuth` group, all methods' throughput is around 100, because they all use password hashing on the server.

For the Internet experiments shown in Figure 5b,



(a) Testbed (b) Internet

Figure 6: **Median CPU time of pre-authentication (`PreAuth`) and full authentication (`FullAuth`) for four methods. "N/A" indicates the absence of corresponding operations. We omit `FullAuth` results on the Internet since they match those on the testbed due to the identical operations and server machine. The numbers above bars indicate the time values.**

PreAcher's `PreAuth` throughput reduces. We find that this reduction originates from the 2-RTT design of PreAcher and `wrk`. Specifically, the 2-RTT design increases the latency, and in `wrk`, each connection waits for a request to complete before sending the next one. As a result, a higher latency leads to a lower throughput when all 64 connections are saturated. In contrast, DuoHash's `PreAuth` throughput improves because the computation is no longer a bottleneck on Cloudflare.

8.4 CPU Time

We measure the CPU Time of the pre-authentication and full authentication operations. Figure 6 shows the median CPU time measured on the testbed and the Internet.

In Figure 6a, both DuoHash and SGX-CDN take much over 100× more CPU time than PreAcher for pre-authentication on the CDN VM, because they both use password hashing while PreAcher does not. Such a difference in CPU times explains the results of ADoS defense and throughput on the testbed in § 8.2 and § 8.3. As for full authentication on the server VM, PreAcher and DuoHash are comparable with the baseline since they all use the same password hashing algorithm.

For the Internet experiments, full authentication results should remain consistent with Figure 6a as we run identical operations on the same server. Therefore, we focus on pre-authentication's CPU time, which is provided by Cloudflare's statistics. As shown in Figure 6b, PreAcher's CPU time enormously increases to 8.3 ms, compared to 0.16 ms on the testbed. This increment arises from the implementation difference: PreAcher adopts C++ for pre-authentication on the testbed but use JavaScript on Cloudflare, as C++ is not supported in its serverless computing. The performance gap reflects the efficiency difference between these two languages.

However, DuoHash's CPU time does not increase on Cloudflare even though it also uses JavaScript. The reason stems from the cryptographic library. For DuoHash, we implement password hashing with Web Crypto API [94], which actually executes native code with little performance downgrade. Since this API does not support the primitive elliptic curve operations required by PreAcher, we adopt a pure JavaScript library libsodium [11] for it, downgrading the performance.

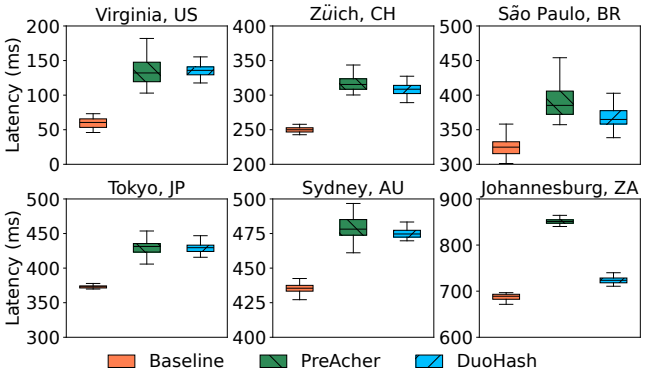


Figure 7: **The latency of a successful login for the clients in six regions around the world.**

Nevertheless, DuoHash still consumes 3× the CPU time of PreAcher on Cloudflare due to password hashing, which will be charged more to websites. Overall, DuoHash is a simple solution, but PreAcher achieves better CPU efficiency. We leave the choice between them for web developers.

8.5 Latency

We measure the end-to-end latency of a successful login for each method. To show a realistic latency, we set up six geographically distributed VMs as clients in six Azure regions covering all continents except Antarctica, including Virginia (US), Zürich (Switzerland), São Paulo (Brazil), Tokyo (Japan), Sydney (Australia), and Johannesburg (South Africa). We set up the origin server in Virginia (US). In each region, we send 100 valid login requests to Cloudflare edge servers. The edge servers will run deployed pre-authentication and request the origin server for full authentication. We define the latency as the time from starting the request to receiving the login success indication.

As shown in Figure 7, in all regions, PreAcher and DuoHash has a higher latency than the baseline. The latency overhead of PreAcher comes from its 2-RTT design in the protocol. However, the additional RTT happens between the client and the CDN, and thus it should not introduce much overhead because the clients are usually close to a CDN's edge server. We spot that PreAcher introduces a high latency overhead in Johannesburg, South Africa. This is because Cloudflare redirects the client's request to an edge server far from Johannesburg, leading to about 140 ms inflation. Except for this region, the overhead of PreAcher's median latency ranges from 42 ms to 72 ms in all other regions, compared to the baseline. Besides, DuoHash introduces overheads ranging from 34 ms to 75 ms to the median latency compared to the baseline in all regions, which is close to PreAcher. The latency overheads of PreAcher and DuoHash primarily stem from their computation time and the access delay of Cloudflare's storage. Since the login process is usually a one-time cost for users during web browsing, we consider the overhead of PreAcher and DuoHash to be acceptable.

9 Discussion

ADoS attacks through other interfaces. Although PreAcher avoids the ADoS attacks exploiting the login interface, other interfaces of a website may still be vulnerable. If the exploited interface requires authentication, the website can track and block accounts that initiate attacks. If the interface is open to public, the website can enforce authentication to remove the attack surface. Websites can also adopt the general ADoS defense proposed in prior studies [44, 67, 73, 90].

Password reuse and weak passwords. A user may reuse exactly the same password across websites. PreAcher cannot prevent account cracking caused by password reuse, which is an inherent limitation of password authentication. To mitigate password reuse, websites can adopt existing password reuse prevention [92] and detection [84] methods, which are compatible with PreAcher.

Similarly, a user may use a common or weak password, and an attacker inside a CDN could collect these passwords and enumerate them offline to guess the user's password. Nevertheless, PreAcher still resists such attacks. Since those common or weak passwords also share similarities, the LSH in PreAcher will mitigate the attacks as discussed in § 5. Additionally, a website can prevent users from using these common or weak passwords during registration.

Alternative solution. An alternative solution to achieve ADoS mitigation and password protection simultaneously is to simply use Trusted Execution Environment (TEE), such as Intel SGX [40]. We evaluate this solution in § 8. However, as TEE requires both hardware and software upgrades and is not enabled by CDNs yet, it is difficult to deploy such a solution. In contrast, PreAcher does not require upgrading of CDNs and can be deployed by websites unilaterally.

10 Related Work

DoS protection. In the past decades, researchers have explored plenty of DDoS defense mechanisms by using the network capabilities [70, 103, 106], the cloud defense [50, 71, 108], the routing polices [86], and the new Internet architecture [97]. In practice, websites widely adopt the DDoS protection offered by a CDN provider or a cloud provider [4, 7, 15]. We do not innovate DDoS protection in PreAcher, but we retain the DDoS protection of CDNs in PreAcher.

Protecting ADoS is more difficult than DDoS because the requests are application-specific and the request amount is usually too small to be detected [2]. Prior studies have proposed ADoS defense mechanisms. However, their solutions focus on the ADoS caused by either regular expressions [66] or specific programming languages [43, 73]. Researchers also propose to detect and filter out ADoS traffic by profiling requests' resource usage and learning the pattern of attack traffic [44, 67, 90], which are less effective to login requests due to the consistent payload pattern of a username and a password. In this paper, we show that ADoS attacks exploiting password login are feasible, even for a server hidden behind a CDN (§ 2.1), and our design of PreAcher mitigates such attacks.

User privacy leakage to CDNs and countermeasures. Researchers have spotted the user privacy leakage to third-party CDNs caused by TLS private key sharing for years [35, 68]. A line of research has proposed countermeasures, including certificate delegation [68], Keyless SSL [88], TLS modifications [30, 65, 79], TEE-based CDNs [27, 54, 78, 95], differential privacy [100], homormophic encryption [98], and a new encryption channel for web [69].

Although these proposals are useful for protecting users' sensitive data against an attacker inside a CDN, none of them mitigates ADoS attacks. PreAcher is related but has different goals. It is not meant to protect all private communication between the client and the server. Instead, it focuses on preventing password leakage and addressing ADoS vulnerability caused by current password authentication.

Password authentication. A series of literature has discussed the security of the password authentication, including the password exposure on CDNs [101], password guessing techniques [80, 102], password storage mechanisms [14, 31], new efficient protocols of password authentication [45, 52, 60], the password reuse practice [42, 93] and its defense [84, 92], etc. PreAcher's authentication protocol is inspired by the advance of Password Authenticated Key Exchange (PAKE) protocol [60], which enables authenticated key exchange between a client and a server without cleartext password checking on the server. However, existing PAKE protocols cannot be directly applied to our system, because the origin server must engage directly with the user in each PAKE session which makes DoS attack unavoidable. Thus, we design a novel three-party authentication protocol in PreAcher to securely offload password authentication to a CDN.

To sum up, PreAcher is the first solution that simultaneously prevents ADoS/DDoS attacks, preserves password secrecy, and provides good usability.

11 Conclusion

This paper introduces PreAcher, a comprehensive solution designed to combat the ADoS/DDoS attacks exploiting password authentication and preserve password secrecy. PreAcher incorporates a three-party authentication protocol that allows CDNs to perform password pre-authentication without accessing the actual passwords. We extensively analyze PreAcher's security features and evaluate its performance. Websites can deploy PreAcher independently and immediately, without requiring any modifications to clients or CDNs.

Acknowledgments

We sincerely thank our shepherd Boris Pismenny and the anonymous reviewers for their constructive comments. We are also grateful to Bobby Bhattacharjee, Zonghao Huang, Matthew Lentz, and Bruce Maggs for their insightful feedback. This work is supported in part by NSF awards CNS-2225448, CNS-2226107 and CNS-2310927.

References

[1] Akamai EdgeWorkers. Retrieved in May, 2024 from https://techdocs.akamai.com/edgeworkers/docs/welcome-to-edgeworkers.

[2] Application Layer DDoS Attack. Retrieved in June, 2024 from https://www.cloudflare.com/learning/ddos/application-layer-ddos-attack/.

[3] Auth0 by Okta. Retreived in June, 2024 from https://auth0.com/.

[4] AWS Shield Features. Retrieved in May, 2024 from https://aws.amazon.com/shield/features/.

[5] BrightData - Scraping Browser API. Retrieved in Aug, 2024 from https://brightdata.com/products/scraping-browser.

[6] Cloudflare. Retrieved in June, 2024 from https://www.cloudflare.com/.

[7] Cloudflare DDoS Protection. Retrieved in May, 2024 from https://developers.cloudflare.com/ddos-protection.

[8] Cloudflare Workers. Retrieved in May, 2024 from https://developers.cloudflare.com/workers/.

[9] Compute Service - AWS Lambda. Retrieved in May, 2024 from https://aws.amazon.com/pm/lambda.

[10] DCsv2 sizes series - Azure Virtual Machines. Retrieved in Jan, 2025 from https://learn.microsoft.com/en-us/azure/virtual-machines/sizes/general-purpose/dcsv2-series?tabs=sizebasic.

[11] libsodium. Retrieved in Aug, 2024 from https://doc.libsodium.org/.

[12] OpenSSL. Retrieved in May, 2024 from https://www.openssl.org/.

[13] OpenSSL Documentation: openssl-enc. Retrieved in Sep, 2024 from https://docs.openssl.org/master/man1/openssl-enc/#options.

[14] OWASP Password Storage Cheat Sheet. Retrieved in June, 2024 from https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html.

[15] Prolexic DDoS Protection: Reference Architecture. Retrieved in May, 2024 from https://www.akamai.com/resources/reference-architecture/prolexic-ddos-protection.

[16] Puppeteer. Retrieved in Dec, 2024 from https://www.brightdata.com.

[17] Sogou c++ workflow. Retrieved in May, 2024 from https://github.com/sogou/workflow.

[18] What Is Credential Stuffing? Retreived in Dec, 2024 from https://www.akamai.com/glossary/what-is-credential-stuffing.

[19] What Is Credential Stuffing? Retreived in Dec, 2024 from https://www.cloudflare.com/learning/bots/what-is-credential-stuffing.

[20] wrk2. Retrieved in May, 2024 from https://github.com/giltene/wrk2.

[21] OpenID Authentication 2.0 - Final. https://openid.net/specs/openid-authentication-2_0.html, 2007.

[22] CVE-2020-28873. https://www.cve.org/CVERecord?id=CVE-2020-28873, 2021.

[23] CVE-2022-29700. https://www.cve.org/CVERecord?id=CVE-2022-29700, 2022.

[24] 2024 Data Breach Investigations Report. Technical report, Verizon, 2024.

[25] The Malicious Bot Playbook: Early Warning Signs, and What to Do about Them. Technical report, Cloudflare, 2024.

[26] Millions of Accounts Vulnerable due to Google's OAuth Flaw. https://trufflesecurity.com/blog/millions-at-risk-due-to-google-s-oauth-flaw, 2025.

[27] Rufaida Ahmed, Zirak Zaheer, Richard Li, and Robert Ricci. Harpocrates: Giving Out Your Secrets and Keeping Them Too. In *Proceedings of IEEE/ACM Symposium on Edge Computing (SEC'18)*, pages 103–114. IEEE, 2018.

[28] Suood Al Roomi and Frank Li. A Large-Scale Measurement of Website Login Policies. In *Proceedings of USENIX Security Symposium 2023*, pages 2061–2078. USENIX, 2023.

[29] José Becerra, Peter YA Ryan, Petra Šala, and Marjan Škrobot. An Offline Dictionary Attack against zkPAKE Protocol. In *Proceedings of the 11th ACM Conference on Security & Privacy in Wireless and Mobile Networks (WiSec'18)*, pages 291–292. ACM, 2018.

[30] Karthikeyan Bhargavan, Ioana Boureanu, Antoine Delignat-Lavaud, Pierre-Alain Fouque, and Cristina Onete. A Formal Treatment of Accountable Proxying

over TLS. In *Proceedings of S&P'18*, pages 799–816. IEEE, 2018.

[31] Alex Biryukov, Daniel Dinu, and Dmitry Khovratovich. Argon2: New Generation of Memory-Hard Functions for Password Hashing and Other Applications. In *Proceedings of EuroS&P'16*, pages 292–302. IEEE, 2016.

[32] Alex Biryukov, Daniel Dinu, Dmitry Khovratovich, and Simon Josefsson. Argon2 Memory-Hard Function for Password Hashing and Proof-of-Work Applications. RFC 9106, Internet Engineering Task Force (IETF), 2021.

[33] Joseph Bonneau, Cormac Herley, Paul C Van Oorschot, and Frank Stajano. The Quest to Replace Passwords: A Framework for Comparative Evaluation of Web Authentication Schemes. In *Proceedings of S&P'12*, pages 553–567. IEEE, 2012.

[34] Ran Canetti, Shai Halevi, and Michael Steiner. Mitigating Dictionary Attacks on Password-Protected Local Storage. In *Proceedings of CRYPTO'06*, pages 160–179. Springer, 2006.

[35] Frank Cangialosi, Taejoong Chung, David Choffnes, Dave Levin, Bruce M Maggs, Alan Mislove, and Christo Wilson. Measurement and Analysis of Private Key Sharing in the HTTPS Ecosystem. In *Proceedings of CCS'16*, pages 628–640. ACM, 2016.

[36] Anthony D Carter and Richard A Johnson. Slow Hashing Speed as a Protection for Weak Passwords. *International Journal of Advanced Engineering and Science*, 2020.

[37] Sílvia Casacuberta, Julia Hesse, and Anja Lehmann. SoK: Oblivious Pseudorandom Functions. In *Proceedings of EuroS&P'22*, pages 625–646. IEEE, 2022.

[38] Julio Casal. 1.4 Billion Clear Text Credentials Discovered in a Single Database. https://medium.com/4iqdelvedeep/1-4-billion-clear-text-credentials-discovered-in-a-single-database-3131d0a1ae14, 2017.

[39] Stéphane Ciolino, Simon Parkin, and Paul Dunphy. Of Two Minds About Two-Factor: Understanding Everyday FIDO U2F Usability through Device Comparison and Experience Sampling. In *Proceedings of SOUPS'19*, pages 339–356. USENIX, 2019.

[40] Victor Costan and Srinivas Devadas. Intel SGX Explained. *IACR Cryptology ePrint Archive*, 2016(086):1–118, 2016.

[41] Scott A Crosby and Dan S Wallach. Denial of Service via Algorithmic Complexity Attacks. In *Proceedings of USENIX Security Symposium 2003*. USENIX, 2003.

[42] Anupam Das, Joseph Bonneau, Matthew Caesar, Nikita Borisov, and XiaoFeng Wang. The Tangled Web of Password Reuse. In *Proceedings of NDSS'14*, pages 23–26. ISOC, 2014.

[43] James C Davis, Eric R Williamson, and Dongyoon Lee. A Sense of Time for JavaScript and Node.js: First-Class Timeouts as a Cure for Event Handler Poisoning. In *Proceedings of USENIX Security Symposium 2018*, pages 343–359. USENIX, 2018.

[44] Henri Maxime Demoulin, Isaac Pedisich, Nikos Vasilakis, Vincent Liu, Boon Thau Loo, and Linh Thi Xuan Phan. Detecting Asymmetric Application-layer Denial-of-Service Attacks In-Flight with Finelame. In *Proceedings of ATC'19*, pages 693–708. USENIX, 2019.

[45] Bruno Freitas Dos Santos, Yanqi Gu, Stanislaw Jarecki, and Hugo Krawczyk. Asymmetric PAKE with Low Computation and Communication. In *Proceedings of EUROCRYPT'22*, pages 127–156. Springer, 2022.

[46] Huayi Duan, Rubén Fischer, Jie Lou, Si Liu, David Basin, and Adrian Perrig. RHINE: Robust and High-performance Internet Naming with E2E Authenticity. In *Proceedings of NSDI'23*, pages 531–553. USENIX, 2023.

[47] Christos A Fidas, Artemios G Voyiatzis, and Nikolaos M Avouris. On the Necessity of User-Friendly CAPTCHA. In *Proceedings of CHI'11*, pages 2623–2626. ACM, 2011.

[48] Sanam Ghorbani Lyastani, Sven Bugiel, and Michael Backes. A Systematic Study of the Consistency of Two-Factor Authentication User Journeys on Top-Ranked Websites. In *Proceedings of NDSS'23*. ISOC, 2023.

[49] Omer Gil. Web Cache Deception Attack. https://omergil.blogspot.com/2017/02/web-cache-deception-attack.html, 2017.

[50] Yossi Gilad, Amir Herzberg, Michael Sudkovitch, and Michael Goberman. CDN-on-Demand: An Affordable DDoS Defense via Untrusted Clouds. In *Proceedings of NDSS'16*. ISOC, 2016.

[51] Paul A Grassi, James L Fenton, Elaine M Newton, Ray A Perlner, Andrew R Regenscheid, William E Burr, Justin P Richer, Naomi B Lefkovitz, Jamie M Danker, Y Choong, et al. NIST Special Publication 800-63B: Digital Identity Guidelines. *National Institute of Standards and Technology (NIST)*, 27, 2016.

[52] Yanqi Gu, Stanislaw Jarecki, and Hugo Krawczyk. KHAPE: Asymmetric PAKE from Key-Hiding Key Exchange. In *Proceedings of CRYPTO'21*, pages 701–730. Springer, 2021.

[53] Dick Hardt. The OAuth 2.0 Authorization Framework. RFC 6749, Internet Engineering Task Force (IETF), 2012.

[54] Stephen Herwig, Christina Garman, and Dave Levin. Achieving Keyless CDNs with Conclaves. In *Proceedings of USENIX Security Symposium 2020*, pages 735–751. USENIX, 2020.

[55] Scott Hollier, Jania Sajka, Jason White, and Michael Cooper. Inaccessibility of CAPTCHA. *W3C Recommendation*, 2021.

[56] Zonghao Huang, Lujo Bauer, and Michael K Reiter. The Impact of Exposed Passwords on Honeyword Efficacy. In *Proceedings of USENIX Security Symposium 2024*. USENIX, 2024.

[57] Philip J. Nesser II, Mike Straw, Craig Metz, and Neil M. Haller. A One-Time Password System. RFC 2289, Internet Engineering Task Force (IETF), 1998.

[58] Omid Jafari, Preeti Maurya, Parth Nagarkar, Khandker Mushfiqul Islam, and Chidambaram Crushev. A Survey on Locality Sensitive Hashing Algorithms and Their Applications. *arXiv preprint arXiv:2102.08942*, 2021.

[59] Stanislaw Jarecki, Aggelos Kiayias, and Hugo Krawczyk. Round-Optimal Password-Protected Secret Sharing and T-PAKE in the Password-Only Model. Cryptology ePrint Archive, Paper 2014/650, 2014.

[60] Stanislaw Jarecki, Hugo Krawczyk, and Jiayu Xu. OPAQUE: an Asymmetric PAKE Protocol Secure Against Pre-Computation Attacks. In *Proceedings of EUROCRYPT'18*, pages 456–486. Springer, 2018.

[61] Eric Jonas, Johann Schleier-Smith, Vikram Sreekanti, Chia-Che Tsai, Anurag Khandelwal, Qifan Pu, Vaishaal Shankar, Joao Carreira, Karl Krauth, Neeraja Yadwadkar, Joseph E Gonzalez, Raluca Ada Popa, Ion Stoica, and David A Patterson. Cloud Programming Simplified: A Berkeley View on Serverless Computing. Technical report, University of California, Berkeley, 2019.

[62] Matt Klein. Introduction to Modern Network Load Balancing and Proxying. https://blog.envoyproxy.io/introduction-to-modern-network-load-balancing-and-proxying-a57f6ff80236, 2017.

[63] Hugo Krawczyk, Mihir Bellare, and Ran Canetti. HMAC: Keyed-Hashing for Message Authentication. RFC 2104, Internet Engineering Task Force (IETF), 1997.

[64] Kat Krol, Eleni Philippou, Emiliano De Cristofaro, and M Angela Sasse. "They brought in the horrible key ring thing!" Analysing the Usability of Two-Factor Authentication in UK Online Banking. In *Proceedings of NDSS Workshop on Usable Security*. ISOC, 2015.

[65] Hyunwoo Lee, Zach Smith, Junghwan Lim, Gyeongjae Choi, Selin Chun, Taejoong Chung, and Ted Taekyoung Kwon. maTLS: How to Make TLS Middlebox-aware? In *Proceedings of NDSS'19*. ISOC, 2019.

[66] Yeting Li, Zixuan Chen, Jialun Cao, Zhiwu Xu, Qiancheng Peng, Haiming Chen, Liyuan Chen, and Shing-Chi Cheung. ReDoSHunter: A Combined Static and Dynamic Approach for Regular Expression DoS Detection. In *Proceedings of USENIX Security Symposium 2021*, pages 3847–3864. USENIX, 2021.

[67] Zhi Li, Hai Jin, Deqing Zou, and Bin Yuan. Exploring New Opportunities to Defeat Low-Rate DDoS attack in Container-based Cloud Environment. *IEEE Transactions on Parallel and Distributed Systems (TPDS'19)*, 31(3):695–706, 2019.

[68] Jinjin Liang, Jian Jiang, Haixin Duan, Kang Li, Tao Wan, and Jianping Wu. When HTTPS meets CDN: A Case of Authentication in Delegated Service. In *Proceedings of S&P'14*, pages 67–82. IEEE, 2014.

[69] Shihan Lin, Rui Xin, Aayush Goel, and Xiaowei Yang. InviCloak: An End-to-End Approach to Privacy and Performance in Web Content Distribution. In *Proceedings of CCS'22*, pages 1947–1961. ACM, 2022.

[70] Xin Liu, Xiaowei Yang, and Yong Xia. NetFence: Preventing Internet Denial of Service from Inside out. In *Proceedings of SIGCOMM'10*, pages 255–266. ACM, 2010.

[71] Zhuotao Liu, Hao Jin, Yih-Chun Hu, and Michael Bailey. MiddlePolice: Toward Enforcing Destination-Defined Policies in the Middle of the Internet. In *Proceedings of CCS'16*, pages 1268–1279. ACM, 2016.

[72] Guillaume Marçais, Dan DeBlasio, Prashant Pandey, and Carl Kingsford. Locality-Sensitive Hashing for the Edit Distance. *Bioinformatics*, 35(14):i127–i135, 2019.

[73] Wei Meng, Chenxiong Qian, Shuang Hao, Kevin Borgolte, Giovanni Vigna, Christopher Kruegel, and Wenke Lee. Rampart: Protecting Web Applications from CPU-Exhaustion Denial-of-Service Attacks. In *Proceedings of USENIX Security Symposium 2018*, pages 393–410. USENIX, 2018.

[74] Ralph Charles Merkle. *Secrecy, Authentication, and Public Key Systems*. PhD thesis, Stanford university, Stanford, CA, 1979.

[75] Seyed Ali Mirheidari, Sajjad Arshad, Kaan Onarlioglu, Bruno Crispo, Engin Kirda, and William Robertson. Cached and Confused: Web Cache Deception in the Wild. In *Proceedings of USENIX Security Symposium 2020*, pages 665–682. USENIX, 2020.

[76] Seyed Ali Mirheidari, Matteo Golinelli, Kaan Onarlioglu, Engin Kirda, and Bruno Crispo. Web Cache Deception Escalates! In *Proceedings of USENIX Security Symposium 2022*, pages 179–196. USENIX, 2022.

[77] Kathleen Moriarty, Burt Kaliski, and Andreas Rusch. PKCS# 5: Password-Based Cryptography Specification Version 2.1. RFC 8108, Internet Engineering Task Force (IETF), 2017.

[78] David Naylor, Richard Li, Christos Gkantsidis, Thomas Karagiannis, and Peter Steenkiste. And then There were More: Secure Communication for More than Two Parties. In *Proceedings of CoNEXT'17*, pages 88–100. ACM, 2017.

[79] David Naylor, Kyle Schomp, Matteo Varvello, Ilias Leontiadis, Jeremy Blackburn, Diego R López, Konstantina Papagiannaki, Pablo Rodriguez Rodriguez, and Peter Steenkiste. Multi-Context TLS (mcTLS): Enabling Secure In-Network Functionality in TLS. In *Proceedings of SIGCOMM'15*, pages 199–212. ACM, 2015.

[80] Bijeeta Pal, Tal Daniel, Rahul Chatterjee, and Thomas Ristenpart. Beyond credential stuffing: Password Similarity Models Using Neural Networks. In *Proceedings of S&P'19*, pages 417–434. IEEE, 2019.

[81] Colin Percival and Simon Josefsson. The scrypt Password-Based Key Derivation Function. RFC 7914, Internet Engineering Task Force (IETF), 2016.

[82] Benny Pinkas and Tomas Sander. Securing Passwords Against Dictionary Attacks. In *Proceedings of CCS'02*, pages 161–170. ACM, 2002.

[83] Will Reese. NGINX: the High-Performance Web Server and Reverse Proxy. *Linux Journal*, 2008(173):2, 2008.

[84] Stuart Schechter, Cormac Herley, and Michael Mitzenmacher. Popularity is Everything: A New Approach to Protecting Passwords from Statistical-Guessing Attacks. In *Proceedings of HotSec'10*. USENIX, 2010.

[85] Youren Shen, Hongliang Tian, Yu Chen, Kang Chen, Runji Wang, Yi Xu, Yubin Xia, and Shoumeng Yan. Occlum: Secure and Efficient Multitasking inside a Single Enclave of Intel SGX. In *Proceedings of ASPLOS'20*, pages 955–970. ACM, 2020.

[86] Jared M Smith and Max Schuchard. Routing Around Congestion: Defeating DDoS Attacks and Adverse Network Conditions via Reactive BGP Routing. In *Proceedings of S&P'18*, pages 599–617. IEEE, 2018.

[87] Robert Snelick, Umut Uludag, Alan Mink, Mike Indovina, and Anil Jain. Large-scale Evaluation of Multimodal Biometric Authentication Using State-of-the-art Systems. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI'05)*, 27(3):450–455, 2005.

[88] Nick Sullivan. Keyless SSL: The Nitty Gritty Technical Details. https://blog.cloudflare.com/keyless-ssl-the-nitty-gritty-technical-details/, 2014.

[89] San-Tsai Sun, Eric Pospisil, Ildar Muslukhov, Nuray Dindar, Kirstie Hawkey, and Konstantin Beznosov. What makes Users Refuse Web Single Sign-On? An Empirical Investigation of OpenID. In *Proceedings of SOUPS'11*, pages 1–20. USENIX, 2011.

[90] Rajat Tandon, Haoda Wang, Nicolaas Weideman, Shushan Arakelyan, Genevieve Bartlett, Christophe Hauser, and Jelena Mirkovic. Leader: Defense Against Exploit-Based Denial-of-Service Attacks on Web Applications. In *Proceedings of International Symposium on Research in Attacks, Intrusions and Defenses (RAID'23)*, pages 744–758. ACM, 2023.

[91] Ding Wang, Zijian Zhang, Ping Wang, Jeff Yan, and Xinyi Huang. Targeted Online Password guessing: An Underestimated Threat, 2016.

[92] Ke Coby Wang and Michael K Reiter. How to End Password Reuse on the Web. In *Proceedings of NDSS'19*. ISOC, 2019.

[93] Rick Wash, Emilee Rader, Ruthie Berman, and Zac Wellmer. Understanding Password Choices: How Frequently Entered Passwords are Re-Used Across Websites. In *Proceedings of SOUPS'16*, pages 175–188. USENIX, 2016.

[94] Mark Watson. Web Cryptography API. *W3C Recommendation*, 2017.

[95] Changzheng Wei, Jian Li, Weigang Li, Ping Yu, and Haibing Guan. STYX: A Trusted and Accelerated Hierarchical SSL Key Management and Distribution System for Cloud Based CDN Application. In *Proceedings of SoCC'17*, pages 201–213. ACM, 2017.

[96] Wei Wu, Bin Li, Ling Chen, Junbin Gao, and Chengqi Zhang. A Review for Weighted Minhash Algorithms. *IEEE Transactions on Knowledge and Data Engineering (TKDE'20)*, 34(6):2553–2573, 2020.

[97] Marc Wyss and Adrian Perrig. Zero-setup Intermediate-rate Communication Guarantees in a Global Internet. In *Proceedings of USENIX Security Symposium 2024*, pages 253–270. USENIX, 2024.

[98] Yunming Xiao, Yanqi Gu, Yibo Zhao, Sen Lin, and Aleksandar Kuzmanovic. Enabling Anonymous Online Streaming Analytics at the Network Edge. *ACM Transactions on Computer Systems (TOCS'25)*, 2025.

[99] Yunming Xiao and Matteo Varvello. FIAT: Frictionless Authentication of IoT Traffic. In *Proceedings of CoNEXT'22*, pages 156–170. ACM, 2022.

[100] Yunming Xiao, Yibo Zhao, Sen Lin, and Aleksandar Kuzmanovic. Snatch: Online Streaming Analytics at the Network Edge. In *Proceedings of EuroSys'24*, pages 349–369. ACM, 2024.

[101] Rui Xin, Shihan Lin, and Xiaowei Yang. Quantifying User Password Exposure to Third-Party CDNs. In *Proceedings of PAM'23*, pages 652–668. Springer, 2023.

[102] Ming Xu, Jitao Yu, Xinyi Zhang, Chuanwang Wang, Shenghao Zhang, Haoqi Wu, and Weili Han. Improving Real-world Password Guessing Attacks via Bi-Directional Transformers. In *Proceedings of USENIX Security Symposium 2023*, pages 1001–1018. USENIX, 2023.

[103] Xiaowei Yang, David Wetherall, and Thomas Anderson. A DoS-Limiting Network Architecture. In *Proceedings of SIGCOMM'05*, pages 241–252. ACM, 2005.

[104] Zhuolin Yang, Yuxin Chen, Zain Sarwar, Hadleigh Schwartz, Ben Y Zhao, and Haitao Zheng. Towards a General Video-based Keystroke Inference Attack. In *Proceedings of USENIX Security Symposium 2023*, pages 141–158. USENIX, 2023.

[105] Xiaokuan Zhang, Xueqiang Wang, Xiaolong Bai, Yinqian Zhang, and XiaoFeng Wang. OS-Level Side Channels without Procfs: Exploring Cross-App Information Leakage on IOS. In *Proceedings of NDSS'18*. ISOC, 2018.

[106] Xin Zhang, Hsu-Chun Hsiao, Geoffrey Hasker, Haowen Chan, Adrian Perrig, and David G Andersen. SCION: Scalability, Control, and Isolation on Next-Generation Networks. In *Proceedings of S&P'11*, pages 212–227. IEEE, 2011.

[107] Zirui Neil Zhao, Adam Morrison, Christopher W Fletcher, and Josep Torrellas. Last-Level Cache Side-Channel Attacks Are Feasible in the Modern Public Cloud. In *Proceedings of ASPLOS'24*, pages 582–600. ACM, 2024.

[108] Shengbao Zheng and Xiaowei Yang. DynaShield: Reducing the Cost of DDoS Defense using Cloud Services. In *Proceedings of ATC Workshop on Hot Topics in Cloud Computing*. USENIX, 2019.

A PreAcher Registration Protocol

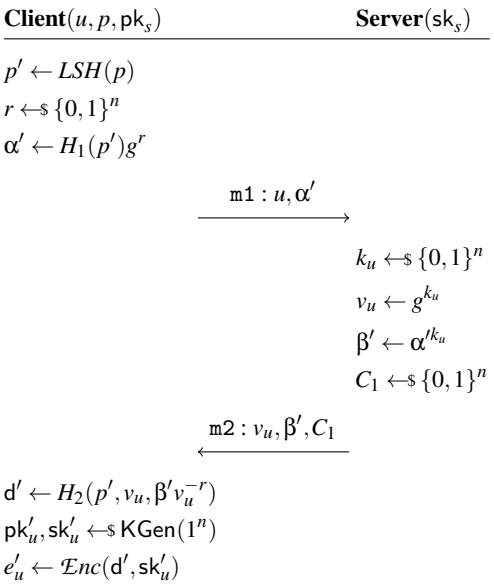


Figure 8: **First round of PreAcher registration.**

In the registration, the client communicates to the server through a CDN. The CDN simply forwards the messages and is not involved in the protocol. As we discuss in § 2.3, the registration can be considered as ADoS-resist in reality, but the CDN can observe the message content. At the end of the registration, the server will send some registration information without exposing user passwords to the CDN so that the CDN can use such information for pre-authentication. The registration protocol works as follows:

Initialization. A user inputs the username u and password p to the client. As discussed in 3.1, the client obtains the server's public key pk_s from the HTML of the login page, and the server keeps the corresponding private key sk_s.

Generating a symmetric key from the password. The client and server compute a symmetric key d' together through OPRF. This step combines the OPRF with LSH as shown in Figure 8.

Specifically, the client uses LSH to map p to p'. Then it generates a random nonce r and computes α' used by OPRF. The client sends u and α' to the server for registration (`m1`). The server generates a user-specific k_u and computes v_u, β' from α' and k_u. The server also generates a one-time random challenge C_1 to verify the client's identity in the next round of communication. Then v_u, β', and C_1 is returned to the client (`m2`). Thus, the client computes a secret symmetric key d' through H_2 and d' is hidden from the server. Note that it can be proved that d' is independent from the choice of r.

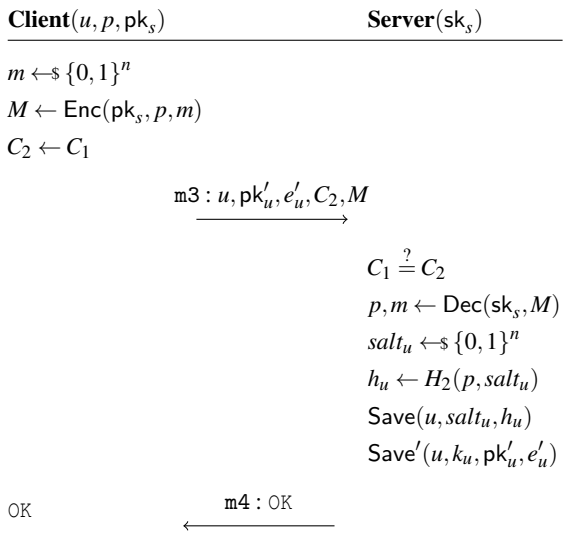


Figure 9: **Second round of PreAcher registration.**

Enclosing a new private key with the symmetric key. The client generates a new key pair $\langle \mathsf{sk}'_u, \mathsf{pk}'_u \rangle$ for the user. Then sk'_u is enclosed into an envelope e'_u by encryption with d'. This new key pair and e'_u will be used for pre-authentication by the CDN in the login.

Registing with the server for full authentication. Finally, the client and the origin server register the user through the traditional password hashing for compatibility with existing authentication in websites. Note that since we trust the origin server (§ 2.3), we can expose the password to the server. Nevertheless, the password sent to the server is encrypted by the server's public key to avoid exposure to the CDN.

As shown in Figure 9, the client uses pk_s to encapsulate p with a random nonce m. The client copies received C_1 as C_2 and sends it back to the server with pk'_u, e'_u, and M (m3). The server first needs to check C_2 against C_1 to identify the client as the one who initiated the registration in the previous round. The server obtains p by decrypting M with its own sk_s. Then it computes the corresponding salted hash value h_u from p. The server saves $salt_u$ and h_u in a user-specific record for future full authentication. Besides, the server saves $k_u, \mathsf{pk}'_u, e_u, e'_u$ and sends them to the CDN for the future pre-authentication ($\mathsf{Save}'(u, k_u, \mathsf{pk}'_u, e'_u)$). Then the server returns the indication of a successful registration (m4) to the client.

B Attack Attempts and Defense

PreAcher can defend against a passive attacker inside the CDN. To make this tangible, we provide two examples of attack attempts to demonstrate the effectiveness of PreAcher's defense.

For instance, an attacker inside the CDN can observe M in $m4$ (Figure 3) during a valid user login. The attacker may attempt to guess the password offline by computing a M value with each guessed password and validating it against the observed M. PreAcher defends against this attack because M is generated by the client using not only the password but also a random nonce m, forcing the attacker to guess m from a vast space in order to obtain a M value that matches the observed one.

Moreover, an attacker may replay the collected M from a valid login and send this M to the server for full authentication. However, the server can prevent such replay attacks by recording the recently received M and thus reject duplicate ones, which is already adopted by prior work [69].

C DuoHash Protocol

We use the same notations described in § 4.1. Figure 10a illustrates the protocol of registration by following steps:

1. The user inputs the username u and password p. Then the client encrypts the password with the server's public key pk_s and sends the username u and encrypted messages M to the server.
2. The origin server decrypts M with its private key sk_s to obtain the password. The server stores a salted hash value h_u for future authentication. Besides, the server calculates another hash value h'_u, which will be used by the CDN for pre-authentication. To avoid offline dictionary attacks by the CDN, the server first uses LSH to map the p to a value p'. Finally, the server uses two layers of hashing H_2 and H_1 to map p' into h'_u, and it is an indication of successful registration to the client.

After the registration, the server sends h'_u and $salt'_u$ to the CDN (not shown in the figure). The CDN stores h'_u and $salt'_u$ for future pre-authentication in the login phase.

The protocol in Figure 10b shows the login phase by following steps:

1. Similar to the registration, the client has u and p and creates the encrypted password M with pk_s. In addition, the client uses LSH to generate p' and the corresponding first-layer hash value h'_1, which is sent to the CDN with u and M.
2. The CDN retrieves $salt'_u$ and h'_u from the user-specific record according to username u. It then conducts the authentication by checking whether $H_2(h', salt'_u)$ is equal to h'_u. If they are equal, the CDN considers the password pre-authentication succeeds and then sends u and M to the origin server for further authentication. If the pre-authentication fails, the CDN directly returns an indication of failed login to the client (not shown in the figure), and thus filters out most failed login requests for the origin server.
3. When the server receives an authentication request from the CDN, it decrypts M with the private key to extract the password p. The server authenticates the user by comparing h_u with the computed $H_2(p, salt_u)$. The server returns the final authentication results to the CDN.
4. The CDN directly forwards back the authentication result to the client.

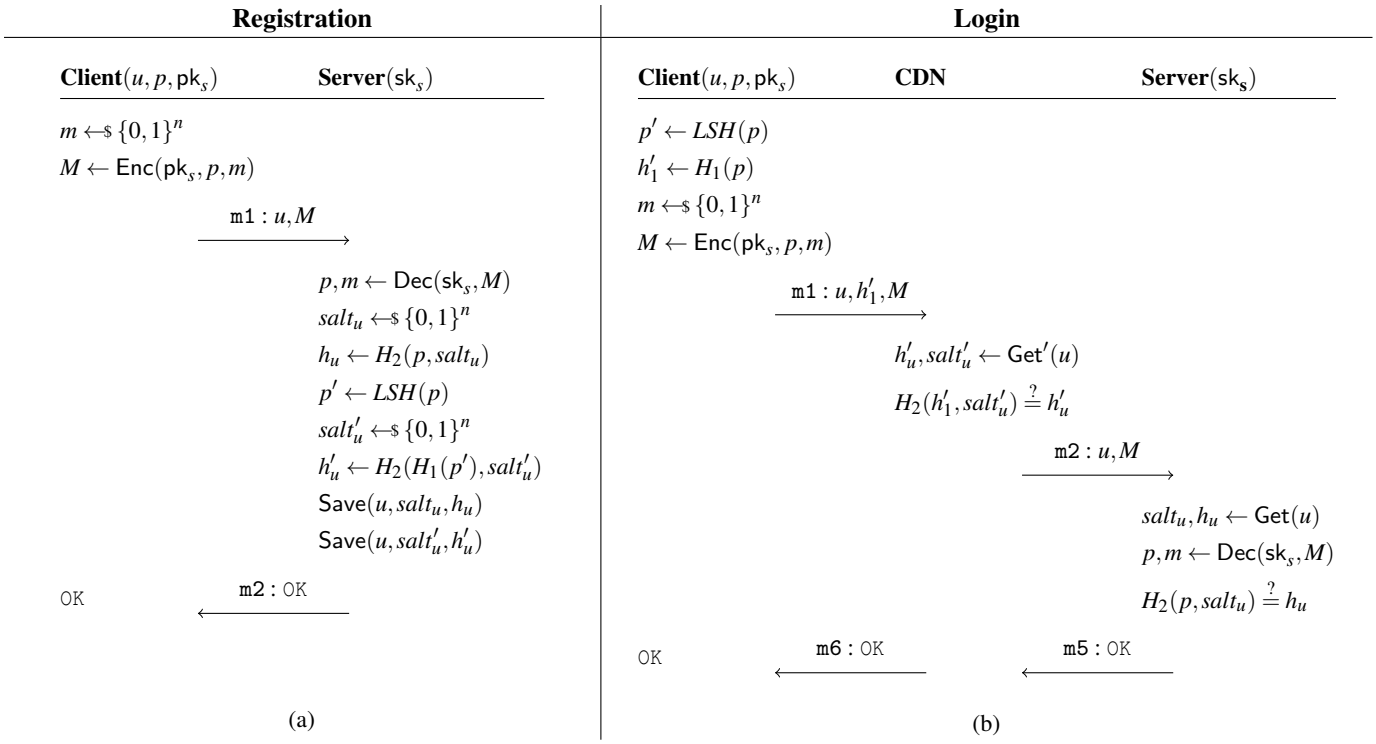


Figure 10: **DuoHash Protocol**

D Security Proof Sketch

In this section, we present a proof sketch to show the security of our protocol. The security requirements are discussed in Section 6. For clarity, we show the complete flow of PreAcher authentication protocol in Figure 11. We use the following probability $\varepsilon_{\mathsf{oprf.random}}$, $\varepsilon_{\mathsf{CPA}}$, $\varepsilon_{\mathsf{CCA}}$, $\varepsilon_{\mathsf{EU-CMA}}$ to denote adversarial advantage of winning the pseudorandomness game of OPRF, the CPA security game, CCA security game, unforgeability game of signature scheme, respectively, which are all negligible in terms of security parameter n.

Theorem 1. *PreAcher is secure against an external attacker, assuming a secure OPRF, a CPA-secure encryption Enc, a CCA-secure encryption* Enc*, and an EU-CMA-secure signature.*

Proof. First, we consider a passive external attacker, and if this attacker provides a wrong password and obtains a p' to authenticate with CDN, there can be two cases:

(1) p' is incorrect. In this case, the attacker gets a random d'^* which is independent from the correct d'. The adversarial advantage of distinguishing this d'^* from d' can be reduced to the adversarial advantage of winning the pseudorandomness game of OPRF[2] and thus is bounded by $\varepsilon_{\mathsf{oprf.random}}$. And since the attacker doesn't know the correct d', the adversarial advantage of distinguishing sk'_u is bounded by the adversarial advantage of winning the semantic security game of the encryption scheme, i.e. $\varepsilon_{\mathsf{CPA}}$. Without knowing the correct sk'_u, the probability of the attacker issuing/forging a valid signature can be reduced to the probability of the attacker winning the unforgeability game of the signature scheme, which is bounded by $\varepsilon_{\mathsf{EU-CMA}}$.

(2) p' is actually correct. This happens because the password guess is close to the correct one and thus gets mapped into the correct p' by LSH with probability $\varepsilon_{\mathsf{LSH}}$. Since p' is correct, the attacker can successfully authenticate with CDN. However, to authenticate with the origin server, the attacker needs to construct a valid adversarial ciphertext M^* from the honest M. This case can be reduced to the CCA security game and thus bounded by ε_{CCA}.

To summarize, the probability of an external adversary breaking the security of PreAcher is negligible. □

Theorem 2. *PreAcher is secure against a passive CDN attacker, assuming a secure OPRF, and a CCA-secure encryption* Enc*.*

Proof. Second, we consider the case of a malicious CDN. Here we no longer consider adversarial authentication to CDN and mainly focus on authentication to the origin server.

Because this malicious CDN holds OPRF key k_u, it can attempt to launch an offline dictionary attack by running OPRF locally multiple times and computing a dictionary D of d's corresponding to its password guesses, and try to decrypt

[2] Our protocol implements a secure OPRF assuming H_2 is a secure hash function, as proved by OPAQUE [60]

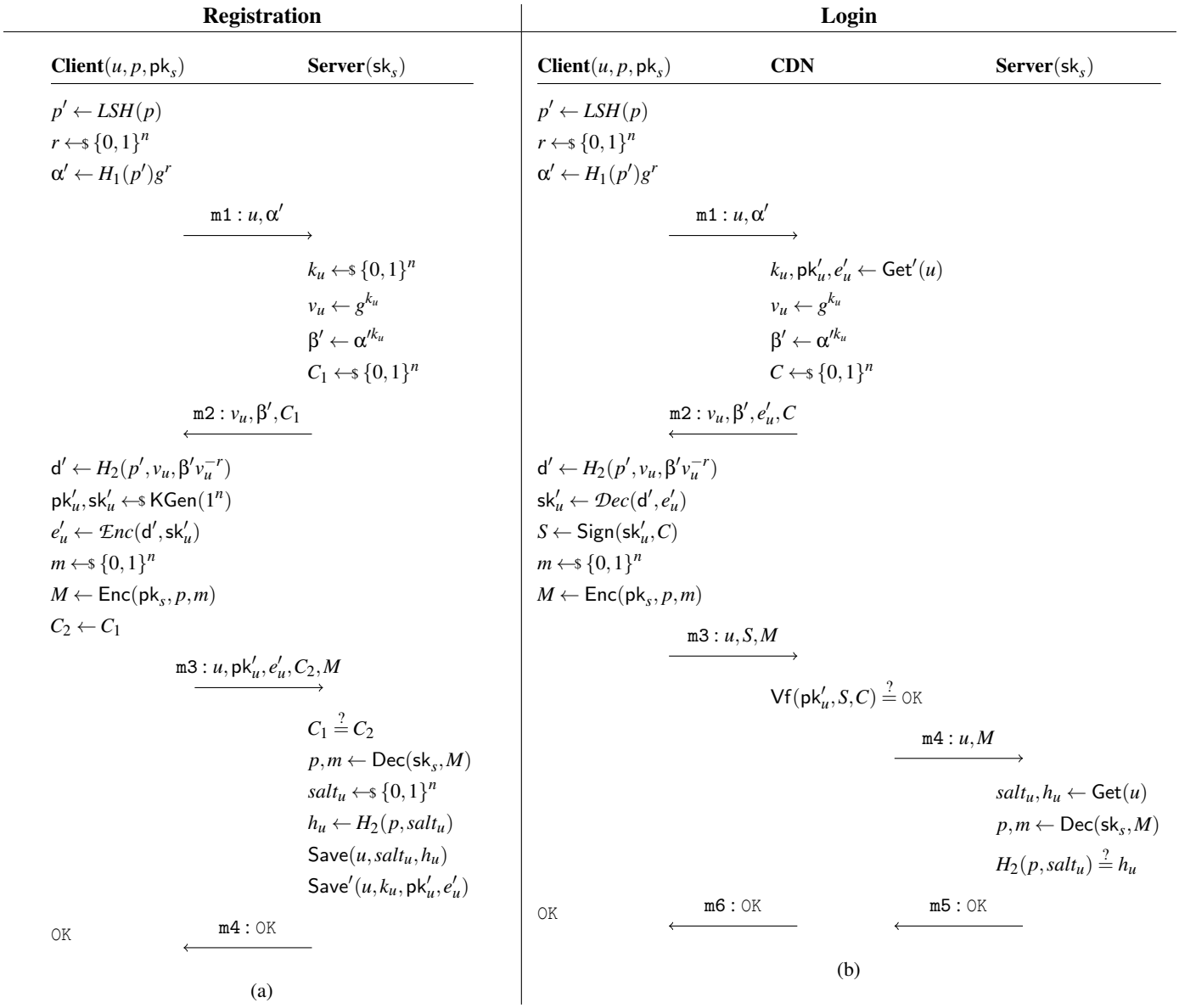


Figure 11: **PreAcher Protocol**

e_u with these d's. However, by the decryption randomness of $\mathcal{D}ec$, any wrong decryption key can only decrypt e_u to a random bitstring, which in the attacker's view is indistinguishable from the correct sk'_u, and the attacker cannot tell from this decryption result whether it uses the correct password or not. The probability of successfully launching such an offline dictionary attack is thus bounded by $\frac{1}{|2^n|} \cdot \frac{1}{|D|}$.

To summarize, the probability of a passive CDN adversary breaking the security of PreAcher is negligible. □