

HTOTP: Honey Time-Based One-Time Passwords

Zixuan Ding^{1b} and Ding Wang^{1b}

Abstract—One-Time Passwords (OTPs) play a crucial role in Two-Factor Authentication (2FA) and Multi-Factor Authentication (MFA) by adding an additional layer of security. OTPs effectively reduce the risk of static passwords being intercepted and reused. Nevertheless, both academic schemes and industrial solutions face security threats stemming from server/device compromises and OTP factor forgery. Chain-based asymmetric OTP schemes are a promising approach to addressing the problem of server compromise but still face threats from device compromise and pre-generated chain leakage. We emphasize that since devices directly store the OTP seed, OTP authentication is essentially equivalent to verifying device possession. This means that in existing OTP schemes, OTP forgery and device compromise remain prevalent and difficult to overcome. In this work, we propose a brand new scheme to address OTP factor forgery and server/device compromises. *For the first time*, our scheme constructs a tightly coupled architecture between the password factor and the OTP factor. The OTP seed is derived from a password and a device-stored salt, preventing OTP seed extraction and OTP forgery even in the event of a device compromise. Through the integration of “honeywords” with the tightly coupled OTP architecture, the server stores decoy OTP seeds generated by decoy passwords, providing resistance against server compromises and partial password guessing from devices. We conduct a comprehensive evaluation of our OTP schemes. The computational overhead is correlated with the number of honeywords, and with the recommended set size of 20, the total verification overhead is approximately 0.24 ms. Additionally, we propose formal security properties and application metrics, and rigorously prove our scheme’s resistance against server/device compromise attacks and guessing attacks. Our scheme is the first to achieve comprehensive OTP security with low overhead.

Index Terms—One-time passwords (OTP), honeywords, two-factor authentication, two-factor security.

I. INTRODUCTION

PASSWORD authentication, long relied upon for its simplicity, faces security challenges due to users’ difficulty in remembering complex or multiple passwords. Human-chosen passwords follow certain patterns [1], and Wang et al.’s research [2] further reveals that the entropy of passwords typically ranges from 20 to 22 bits, with the distribution adhering to Zipf’s law. Moreover, password reuse and similarity heighten the risk of credential stuffing attacks [3]

Received 24 August 2024; revised 6 February 2025; accepted 1 March 2025. Date of publication 11 March 2025; date of current version 2 May 2025. This work was supported in part by the National Natural Science Foundation of China under Grant 62222208 and Grant 62172240; and in part by the Fundamental Research Funds for the Central Universities, Nankai University, under Grant 63253229. The associate editor coordinating the review of this article and approving it for publication was Dr. Ashok Kumar Das. (Corresponding author: Ding Wang.)

The authors are with the College of Cryptology and Cyber Science and the Key Laboratory of Data and Intelligent System Security, Ministry of Education, Nankai University, Tianjin 300350, China, and also with the Key Laboratory of Cyberspace Security Defense, Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100085, China (e-mail: dingzixuan@mail.nankai.edu.cn; wangding@nankai.edu.cn).

Digital Object Identifier 10.1109/TIFS.2025.3550075

once a password is compromised. One-Time Password (OTP) significantly addresses static passwords’ limitations. OTPs add an extra layer of security by ensuring that each authentication code is unique, unpredictable, and cannot be reused.

Lamport [4] first proposed the OTP scheme, named S/KEY [5]. Due to inherent flaws, such as the risk of OTP leakage from pre-generation and storage, S/KEY [5] faces challenges in achieving widespread adoption in the subsequent digital environment. Subsequently, the HMAC-based One-Time Password (HOTP) algorithm was proposed as a new standard in RFC 4226 [6]. In this algorithm, the server and the client share an OTP seed and a counter, using a one-way hash to generate readable OTPs. The Time-based One-Time Password (TOTP) algorithm replaces the counter in HOTP with a timestamp, addressing the issue of desynchronization attacks. HOTP and TOTP are currently the most widely used. However, following the industrial standards of HOTP [6] and TOTP [7], the server stores the OTP seed in plaintext, allowing an adversary who compromises the server to impersonate the user.

To counter potential server compromise, the strategy shifts from symmetric to asymmetric secret sharing. T/Key [8] improves upon S/KEY [5] by making the user device pre-generates a hash chain according to specifically predefined time intervals, and the server stores the tail node of the chain. During verification, the device computes the OTP for the current time by hashing the seed multiple times, and the server hashes the received OTP and compares it with the stored node. However, asymmetric TOTP has its limitations. For instance, it does not protect against device compromises due to the restorable pre-generated chain, and the OTP generation and verification times are long due to iterative hash operations. There is a need for regular regeneration of OTP chains. Also, as OTPs are generated through iterative hashing, used OTPs can be linked and potentially tracked by adversaries.

In addition to security threats of compromises, we highlight crucial issues that affect all OTP schemes. First, the integration of the OTP factor as an additional security layer is loosely coupled with other factors. This loose coupling allows adversaries to independently compromise each factor, thereby simplifying attacks. Second, the device possession represented by OTPs is vulnerable to forgery, such as channel hijacking and OTP (or OTP seed) leakage. A typical example is Google Authenticator [9] and Microsoft Authenticator [10] using HOTP [6] and TOTP [7] standards. These systems implement OTP as a supplementary authentication factor in a loosely coupled manner but do not resolve the issues of loose coupling and vulnerability to forgery.

A. Motivations and Challenges

Although OTP authentication enhances security for identity verification, numerous unresolved issues persist. These problems not only compromise the security of OTP but also

limit its reliability in applications. In this subsection, we discuss the motivations for proposing a new scheme and the challenges in tackling existing issues.

1) *Emphasizing the Possession of the Second Factor:* The password combined with OTP is a widely adopted Two-Factor Authentication (2FA) method to enhance authentication security, where the OTP represents the possession of the device. However, this “possession” can be easily forged, and the verifier cannot determine whether the user holds the device or the secret has been disclosed. We aim to transform the loosely coupled password and device possession (in the form of OTP) into a tightly coupled architecture. This architecture ensures that the knowledge provided by the password is maintained while highlighting the user’s possession of the device. The challenge lies in constructing a tightly coupled architecture for passwords and OTP without altering the user’s habit in password-OTP authentication, while ensuring that the verifier recognizes the OTP as proof of the user’s possession of the device. The construction of a tightly coupled architecture for 2FA has received little attention but urgently needed in existing OTP academic schemes and industrial solutions, and for the first time, this work makes a construction attempt.

2) *Resistance to Secret Compromises:* In the currently prevalent loosely coupled 2FA framework, the server holds complete knowledge for verifying both factors separately (e.g., in password-OTP 2FA [10], [11], the service provider holds the password hash and the OTP seed). Loosely coupled authentication factors only add an extra security layer and cannot offer stronger protection. This allows an adversary who compromises one factor to separately attack other authentication factors. In addition to server compromises, other forms of secret leaks should also be taken seriously. In HOTP, TOTP, or SMS-based OTP, the service provider distributes OTPs (or OTP seeds) to the device. Due to numerous attacks on SMS [12], [13], [14], it is not recommended for OTP distribution [15]. Meanwhile, HOTP and TOTP involve the server distributing OTP seeds in the form of QR codes [11], making it possible for terminal recording attacks to compromise both the OTP seed and the password simultaneously. Trusting the login terminal poses risks to the user, and it is safer for users to rely solely on their own device.

We aim to construct a tightly coupled architecture for password and OTP. The challenge is to prevent an adversary who compromises the server from compromising all authentication factors. Specifically, for the coupled two factors, the server should not independently possess the knowledge to verify them. Moreover, the task is to modify the OTP seed distribution method without changing password-OTP two-factor authentication method, ensuring that powerful adversaries, such as those intercepting channels or controlling terminals, cannot obtain the secrets. Users only need to trust their own devices, eliminating additional trust assumptions.

3) *Detecting Leaks and Guessing Attacks:* It is challenging for both service providers and users to detect server data breaches, device compromises, and guessing attacks. We combine honeywords [16] with the tightly coupled OTP-password architecture to construct an alert mechanism for server compromises and guessing attacks. Specifically, the server stores both the real OTP seed and decoy OTP seeds. These seeds are generated by combining salt with honeywords.

To ensure security, the server itself cannot distinguish the real OTP seeds unless it interacts with the honeychecker. When an adversary compromises the server and tries to log in with honey seeds, or compromises the device to guess passwords, the honeychecker will detect the breach and guessing attempts, alerting the system to the attacks.

B. Contributions

In this work, we first analyze OTP academic schemes and industrial solutions, highlighting security issues and shortcomings related to secret derivation and leakage in symmetric OTP authentication (e.g., HOTP [6] and TOTP [7]). Asymmetric schemes (e.g., S/key [5] and T/key [8]) have drawbacks such as low usability, especially in terms of unfriendly OTP length, limited lifespan, and extended registration and authentication times. *For the first time*, we identify the issues caused by the loose coupling of OTP with other authentication factors: the server authenticates these factors separately and holds all the knowledge for independent verification, meaning that a breach compromises all factors. Furthermore, OTP cannot be used as a proof of the possession of a device.

We construct a tightly coupled two-factor architecture that combines passwords and OTP. The OTP seed is derived from the password and salt, and the user device stores the salt rather than the OTP seed. Consequently, a correct OTP simultaneously indicates the knowledge of the password and user’s possession of the device without forgery. The combination of passwords and OTP resolves issues such as device-side leaks and identity forgery associated with OTP authentication. Inspired by honeywords, we supplement the OTP seeds stored on the server with decoy OTP seeds generated based on honeywords. Honeywords include decoy passwords generated from the original password as well as popular passwords. Therefore, if an attacker tries to log in by compromising the server, or guess passwords online by compromising the device, alerts will be triggered.

In summary, our contributions are four-fold:

- **Tightly coupled password-OTP two-factor architecture.** We transform the loosely coupled architecture of password and OTP (indicating device possession) into a tightly coupled one. The coupling is achieved by converting the two independent factors into a mechanism where the OTP seed is generated from a salt stored on the device and the user’s password together. The combination of device storage and user password knowledge reinforces the user’s possession of the device. The tightly coupled architecture not only prevents adversaries from simultaneously compromising both authentication factors, but also prevents adversaries from separately compromising each factor or gaining access to both factors in sequence through a breach. The tightly coupled architecture addresses issues of factor compromise and impersonation inherent in loosely coupled architectures.
- **Unforgeability and loss resistance of the second factor.** The device receives OTPs from the service provider or stores OTP seeds. The user’s possession of the device signifies possession of the second factor. However, this possession is susceptible to forgery. We change the approach in previous schemes (including standards [5], [6], [7] and popular schemes [8], [17]) to store only a salt instead of the entire OTP seed on the device. In our

construction, the OTP seed is generated by combining the salt with the password. An adversary who compromises the salt without the password cannot generate correct OTPs, thus preventing the adversary from impersonating the user after compromising the device. Additionally, the salt stored on the device is easy to back up, and the compromise of the salt is far less harmful than compromising an OTP seed. Converting the stored OTP seed to a separate salt improves the second factor's resistance to forgery and protects against device-side compromises.

- **Resistance to server compromises and password guessing.** We integrate honeywords into the tightly coupled password-OTP architecture to create a decoy OTP seed mechanism that defends against server compromises and (partial) password guessing attacks. The server stores decoy seeds generated from decoy passwords, alongside the real OTP seed. Since the salt is stored independently on the device, the salted hash stored on the server cannot be used for offline password guessing. Logging in with compromised OTP seeds or guessing the password online post-device compromise will trigger a honeychecker alert. The honey OTP mechanism mitigates password guessing attacks resulting from server breaches and partial¹ guessing attacks originating from the device.
- **Some insights.** The entropy of human-chosen passwords is 20-22 bits, while the entropy of a 6-digit OTP is 19.93 bits, and that of an 8-digit OTP is 26.58 bits. Converting a password into a 6-digit OTP does not significantly reduce entropy in practical authentication scenarios, and thus does not lower the collision difficulty. Variable OTP lengths balance security and user-friendliness. In contrast to the original honeyword mechanism where the server stores salts and salted hashes, storing salts on the device prevents server-compromised adversaries from brute-forcing hashes and semantically recognizing the real password. Meanwhile, incorporating popular passwords into decoy passwords is effective in defending against online guessing attacks from the device. In our honey OTP mechanism, it can achieve a flatness of $1/k$, with k being the number of OTP seeds.

II. PRELIMINARIES AND RELATED WORKS

A. Preliminaries

1) *Honeywords*: Honeywords is a method that enhances the security of password-based authentication systems by mixing multiple decoy passwords [16]. The system stores both the real and decoy passwords for an account, and the honeychecker stores the index of the real password. As a result, it becomes challenging for an adversary who compromises the server to distinguish between the real and decoy passwords. During authentication, the honeychecker verifies the validity of the password index. If an adversary uses a decoy password, the

¹The server stores decoy seeds generated from decoy passwords to alert against online guessing attacks from devices. Decoy passwords are typically composed of popular passwords and variations of the real password. We describe online guessing attacks from the device as partially resisted because the number of decoy passwords is limited.

TABLE I
TERMS AND DEFINITIONS RELATED TO HONEYWORDS

Terms	Definitions
Honeywords	Decoy passwords excluding the real password
Sweetwords	The set of decoy and real passwords
Index	Indexing the real password in the set
Honeychecker	Storing valid index & verifying candidate indexes

honeychecker detects the discrepancy and triggers an alert. Next, we introduce the implementation steps of honeywords and the variant applied in this work.

Assume the i -th user in the authentication system is denoted as u_i , with the password being pw_i . The authentication server maintains a verification table as $(u_i, H(pw_i))$, which generally includes the username and the hashed password (or salted hash $H(pw_i, r)$; the server holds the salt r).

For the password pw_i , the system generates honeywords. The honeywords and the real password pw_i form the sweetwords list $\mathbb{W}_i = (pw_{i,1}, pw_{i,2}, \dots, pw_{i,k})$, where k is the number of elements in the list. Let I_c denote the index of the real password in $\mathbb{W}_i$, that is $pw_{i,I_c} = pw_i$. The correct password is called sugarword, and the remaining $k - 1$ passwords in $\mathbb{W}_i$ are referred to as honeywords. The server stores the hashed sweetwords list $\mathbb{W}_i$ as $(u_i, \mathbb{H}_i)$, where for the elements in $\mathbb{H}_i$, $h_{i,j} = H(pw_{i,j})$. Flatness is used to measure the expected probability that an adversary can guess the real secret from the candidate set. When the probability is $1/k$, it indicates that the elements of the candidate set are flat.

The honeychecker is set up as an auxiliary server that stores the index I_c . Its purpose is to ensure that unless an adversary compromises both servers simultaneously, it is impossible to know which sweetword hash corresponds to the real password. Compromising both servers simultaneously is challenging for the adversary. Note that in the worst-case scenario, if the honeywords system is compromised, password security will only degrade to the level without honeywords. The brief terms and definitions are shown in Table I.

We make the following adjustments to the honeywords architecture to suit this work. The salted hashes stored on the server are no longer used for password authentication but rather as seeds for generating OTPs. Additionally, the salt is stored on the device side instead of the server. In this case, even if an adversary compromises the honeywords system in this work, she can obtain the OTP seed but does not have an advantage in offline password guessing.

2) *Time-Based One-Time Password (TOTP)*: TOTP is widely used in 2FA, generating a temporary password using the current time and a shared secret key [7]. OTPs are generated based on fixed time intervals (e.g., 30 seconds) and the HMAC-SHA-1 algorithm. This method enhances security by providing an additional security layer beyond passwords and is convenient, as OTPs can be generated on various devices. However, OTPs require accurate time synchronization and are valid only for a short period. TOTP generation function is represented as follows: $TOTP(K, T) = Truncate(HMAC-SHA-1(K, T))$, where K is the pre-shared secret, $T = (T_c - T_0)/X$, X is the time step, T_c and T_0 are the current time and initial time, respectively. The output of $HMAC-SHA-1$ [18] is truncated into a user-friendly value (e.g., a 6-digit number).

B. Related Works of OTP

In 1981, Lamport first proposed the scheme [4] of constructing One-Time Passwords (OTPs) based on one-way hash functions. The terminal pre-stores the result of 1000 iterations of hash computations for authentication. In 1995, Lamport's scheme was standardized as the S/KEY OTP system [5] and adopted as an Internet standard [19]. S/KEY suffers from multiple drawbacks. First, the quantity of available OTPs is constrained. Unused OTPs are valid for a long time, leaving them exposed to the risk of being stolen. Additionally, the S/KEY scheme is vulnerable to a "small n " attack [20]. In such an attack scenario, an attacker can impersonate a server to trick the client into leaking unused OTPs. Moreover, the hash function within the S/KEY hash chain is static, making it vulnerable to pre-computation attacks.

Eldefrawy et al. [21] proposed a scheme that constructs two nested hash chains, one for seed updates and the other for OTP generation. The use of multiple hash chains overcomes the limited lifespan of hash chains in S/KEY [4], thus eliminating the necessity for resetting. In Eldefrawy et al.'s scheme, OTP generation and verification involve 4 rounds of challenge-response interactions. Gong et al. [22] constructed an OTP verification scheme based on sub-passwords. During the registration phase, the device stores the lists of random permutation functions and sub-passwords generated by the server, and the verification involves 4 interaction rounds. Park [23] proposed an OTP scheme using multiple short hash chains to address the limitations of hash chain length and OTP generation in S/KEY. However, the seeds for multiple chains still require a pre-shared secret and a counter for generation. Each OTP consists of three different digits and hash values, rendering the scheme less user-friendly compared to short OTPs. Moreover, this scheme still requires online interaction.

In 2005, HMAC-based One-Time Password (HOTP) algorithm was published as RFC 4226 [6]. The HOTP algorithm provides an authentication method where the client and server share a secret and a counter to symmetrically generate human-readable OTPs. The counter increments after each OTP generation, introducing a potential risk of losing synchronization. Time-based One-Time Password (TOTP) algorithm [7] is an improvement on HOTP. TOTP replaces the counter in HOTP with the current timestamp, and the client and server must keep their clocks synchronized. By using the timestamp, TOTP avoids counter desynchronization in HOTP.

There is an asymmetric TOTP method named T/Key [8]. The idea is based on the hash chain structure of Lamport's scheme (S/Key) [4], but each chain node consists of a timestamp and a salt. The salt ensures that each node in the chain produces unique hash values. The device generates a time seed and then builds a hash chain, and the tail node of the chain is stored by the server. For verification, the device sends the node corresponding to the current timestamp to the server, and then the server calculates the hash from the received node to the tail node to verify. Adversaries who try to crack the asymmetric TOTP public key cannot infer any unused passwords. The chain's lifespan is limited, and excessively long chains can lead to uncontrollable verification times. Additionally, the device must regenerate a new chain and interact with the server regularly to initialize the chain.

Yang et al. proposed a scheme called Group TOTP (GTOTP) [24], which allows users to prove their group membership without revealing their real identities. The scheme can transform an asymmetric TOTP scheme into a GTOTP scheme based on permutation schemes and Merkle tree. The root of the Merkle tree serves as the group's public key, and the leaf nodes bind the verification points of group members. Anonymity is achieved through the randomization of the leaf nodes. However, the GTOTP structure has limitations in dynamic group management. Additionally, members in the group incur significant storage overhead, posing challenges to scalability and efficiency in large-scale deployments.

Cao et al. [25] further improved GTOTP's dynamic group management [24], reducing the storage and computation overhead for group members. As an alternative, the proposed Dynamic GTOTP (DGTOTP) introduces a third-party Registration Authority (RA) to handle part of the storage and computation. RA uses virtual verification points as leaves to create a Merkle tree, and then dynamically issues independent secret seeds to joining group members. Each member's secret seed serves as the key for the chameleon hash, allowing the real verification points of the TOTP instances to be dynamically bound to the Merkle tree leaves during authentication. Similarly, DGTOTP still maintains a high overhead.

Erdem and Sandikkaya proposed an OTP service called OTPaaS [17], which introduces an additional OTP provider between the user and the server. The user first registers with the OTP provider, which stores the pairing table of the user, service provider, and OTP provider. OTPaaS transforms the authentication process into a three-party interaction, requiring five rounds of communication. However, the password verification is independent, and OTP as the second factor cannot mitigate the threat to the password if the server is compromised. Additionally, the service provider must trust the OTP provider, and the uncertainty and trust costs associated with the third party are also challenges that hinder implementation.

Sun et al. proposed an OTP strengthening scheme called TrustOTP [26] based on additional hardware ARM TrustZone. All code and data in TrustOTP are securely isolated from the mobile operating system running in the normal domain. This secure isolation ensures that actions in the operating system do not compromise the confidentiality and integrity of the generated OTP or seeds. The OTP or seeds can only be accessed in the secure domain.

Peeters et al. proposed a SMS OTP Security (SOS) scheme [27], aiming to improve the security of SMS-based OTP against message rerouting attacks via Signaling System 7 (SS7) and SIM Swap attacks. During the registration phase, the server and the user's device agree a session key based on the DH key agreement protocol and store the derived keys. During authentication, the server sends a random, 256-bit string to the device via SMS. The device generates an OTP based on the string and the pre-shared key, and returns the OTP to the server for verification through the client. Even if an adversary intercepts the SMS, they cannot generate the correct OTP.

A representative hardware-based OTP service provider is Yubico [28], with its OTP server adopted by companies such as Okta [29], Ping Identity [30], and Secret Double Octopus [31]. Yubico OTP is a 44-character long secret encrypted by a hardware device and verified by a central server, YubiCloud.

The verification server pre-stores the user ID and the AES keys, and maintains a counter of OTPs. The server verifies OTPs by decrypting them. YubiKey also supports OATH-HOTP/TOTP [32], providing feedback for the HOTP token when touched. However, since YubiKey does not have a built-in clock and power, an application (Yubico Authenticator) needs to be installed for TOTP authentication. It may not be practical for users to carry an additional hardware device. In addition, if the device lacks strong security measures, the hardware is vulnerable to attacks if compromised.

In HOTP [6], the user device and the server pre-share a counter for generating OTPs, which is incremented accordingly after each generation. However, if the counters lose synchronization (i.e., exceeding the look-ahead window which is usually set to 5), the OTPs will not match unless the counters are reset. In terms of specific implementation, HOTP [6] is suitable for scenarios involving specific hardware devices and low-frequency authentication requirements. TOTP [7] is widely adopted in mobile applications and online services due to its simplicity and high security derived from time-based synchronization mechanisms. Google Authenticator [9], [33] and Microsoft Authenticator [10] are popular software-based authenticators. These applications use both HOTP [6] and TOTP [7] algorithms to generate correct OTPs, even when the mobile device is offline. During registration, the server generates a key for the user's device, which the user saves by scanning a screen-displayed QR code.

Additionally, there are implementations that further enhance HOTP and TOTP. Authy [34] employs the same OTP generation method as Google Authenticator and offers device synchronization and backup features, enabling users to utilize the same OTP across multiple devices. LastPass Authenticator [35] and 1Password [36] integrate TOTP with password managers, enhancing password management services. Besides TOTP, Duo Mobile [37] further introduces push notification-based verification. Duo Push streamlines the verification process by sending real-time notifications to the user's device, where users simply accept or reject the login request with a single tap.

Above implementations use the OTP-generating/receiving device as a second factor for authentication. However, the second factor and other factors are independent and loosely coupled. An adversary who compromises the server can potentially access both the user's password and the OTP seed. Device loss, secret leakage, and server compromise are all devastating to OTP security. Additionally, untrusted login terminals pose a threat to usernames and passwords, and the leakage of users' long-term keys is extremely dangerous.

C. Related Works of Honeywords

Juels and Rivest first proposed a method named honeywords at CCS'13 to detect password leaks [16]. Honeywords, as decoy passwords, are stored on the server together with the users' real passwords. The index of the real password is stored on an additional server, named honeychecker. The server cannot distinguish the real password from the honeywords and can only interact with the honeychecker to verify the password. An adversary who compromises the server must distinguish the real password from the other $k-1$ honeywords (e.g., the recommended $k = 20$ [16]). Attempting to log in

with honeywords indicates a server compromise, triggering an alert from the honeychecker.

Juels and Rivest [16] divide honeyword generation methods into two categories: methods based on the legacy User Interface (UI) and methods based on the modified UI. The former does not require modifying the client or user behavior, while the latter requires the user to input a password that meets specific requirements. However, the cost of changing behavior is usually high. Chakraborty and Mondal [38] point out that the generation of honeywords by Juels-Rivest is based on random substitution and essentially cannot resist semantic-aware adversaries. Erguler [39] presented a heuristic method named "Honeyindex", which utilizes the real passwords of other users as honeywords.

Wang et al. [40] use empirical experiments to reveal that four honeyword generation methods of Juels-Rivest [16] fail to reach the claimed security. Wang et al. suggest combining different generation models together to overcome the shortcomings of each model. Additionally, Akshima et al. [41] point out that the honeyword generation methods of Juels-Rivest [16] and Chakraborty-Mondal [38] cannot achieve the purported security level. Then, Akshima et al. proposed three methods for legacy UI and modified UI.

Wang et al. [3] point out that there are security and deployment issues in "Honeyindex" [39] and the two legacy UI methods of Akshima et al. [41]. Then, Wang et al. [3] proposed a series of theoretical models based on various capabilities available to the adversary. These models use real-world data sets to design effective experiments to evaluate the flatness of the generation methods, thereby addressing the unresolved issues left by Juels and Rivest [16]. In addition, Wang et al. [3] proposed four honeywords generation methods based on the existing probabilistic password cracking models, and provided a method to re-adjust the cracking model to construct flat honeywords to adapt to future model fusions and deployments.

Research on honeywords provides guidance for generation and application. Li et al. [42] applied honeywords to Password-Authenticated Key Exchange (PAKE) based on the well-known OPAQUE [43] of asymmetric PAKE (aPAKE), and proposed the concept of Honey PAKE (HPAKE). For external adversaries, HPAKE realizes protection based on honeywords. For internal adversaries, the principle of HPAKE is similar to that of aPAKE, and the server stores the one-way value of the password instead of plaintext.

III. SECURITY MODEL

A. Evaluation Criteria

Wang and Wang [44] propose an evaluation set for 2FA schemes in general environments. Erdem and Sandikkaya [17] then extend these criteria to OTP schemes in cloud environments. However, we find that the updated evaluation criteria are still not suitable for general OTP schemes. Therefore, we propose evaluation criteria suitable for the widely used and popular OTP schemes.

- S1. **Resistance to device loss:** The device serves as an authentication factor and is used to generate or receive OTPs. Loss or capture of the device means that this factor may be compromised. We aim to ensure that an adversary's acquisition of the device does not allow them to impersonate the user. In other words, it is hard

for an adversary who compromises the device to forge the user's login.

- S2. **Resistance to server compromise:** Generally, the server stores parameters for verifying authentication factors. Server compromise means that the adversary may be able to break the authentication factors offline. Additionally, a server compromise may go undetected, so we aim to ensure that even if the server is compromised, the adversary cannot break all of the authentication factors. Furthermore, it should be tough for an adversary who compromises the server to impersonate a user.
- S3. **Resistance to channel hijacking:** In SMS-based OTP authentication, adversaries can capture OTPs by various means [45]. Therefore, we aim to ensure that even if an adversary hijacks the channel and intercepts the communication, she should not be able to obtain or compute the OTP or other authentication factors.
- S4. **OTP unlinkability:** The correlation between OTPs is unknown to the adversary. That is, even if the adversary masters a large number of used OTPs, she cannot recover the previous or subsequent OTPs.
- S5. **Resistance to replay attacks:** An adversary who intercepts legitimate communication messages or OTPs and resends them later cannot impersonate a legitimate user to gain unauthorized access.
- S6. **Two (multi)-factor security:** Authentication involves two or more factors, and the user cannot successfully log in unless she possesses all the required factors.
- S7. **Secret leakage alert:** Generally, server data compromise can go unnoticed. However, when an adversary attempts to log in using a leaked secret, the service provider can detect the breach and respond by issuing alerts.
- S8. **Guessing attack alert:** Should the second factor (e.g., user devices) be compromised by adversaries, they might launch brute-force password-guessing attempts to gain unauthorized access. In such a case, the server can detect these attempts and respond with alerts.

In addition to security evaluation criteria, we also develop usability evaluation criteria. These criteria allow for a comprehensive evaluation of the OTP system's user experience, ensuring it provides convenient and reliable identity authentication services while maintaining security.

- C1. **No verifier table:** Devices and servers do not store pre-generated OTP lists, as OTPs are generated in real-time for each authentication. This method eliminates the need for a verifier table, reducing both storage space requirements and the risk of leakage.
- C2. **No online interaction:** During the verification phase, both OTP generation and verification processes function offline, eliminating the need for device-server interaction and the demands on equipment and network.
- C3. **Easy to back up and restore:** The device's secret enables effortless backup and restoration on a new device. Additionally, it resists direct exploitation by adversaries. This ensures that users can still log in even if the original device is lost.
- C4. **Password/OTP friendly:** The length and characters of OTPs should balance usability and security, making them user-friendly and convenient for manual input.

- C5. **Resistance to keyboard/screen recording attacks:** OTP schemes should be designed to withstand potential threats at the login terminal, including shoulder surfing, screen logging, and keylogging. This resistance ensures that long-term secrets, such as passwords and OTP seeds, remain secure and are not leaked.
- C6. **No additional expenses:** Beyond registration and verification, there is no additional overhead required for devices and servers to maintain the OTP scheme.
- C7. **No periodic synchronisation:** Post-registration, the device operates with an unlimited lifespan. It eliminates the need for periodic online server interactions to synchronize parameters, thereby ensuring the validity of stored parameters.
- C8. **No additional third party:** Besides the user (with the device) and the service provider, the OTP authentication does not require the participation of additional third parties, that is, the two parties do not need to bear additional trust costs.

Table II shows the comparison between OTP implementations and popular schemes under the above criteria. Our proposed HTOTP has advantages in resisting device loss (S1), server compromise (S2), and subsequent attacks (S7, S8).

B. System Model

In the setup phase, the service provider initializes and publishes public parameters. The user registers the password on a device (e.g., a smartphone), which generates honeywords and calculates corresponding decoy seeds. The device stores the salt and sends the decoy seeds along with the encrypted index of real password to the service provider. In the verification phase, the device generates an OTP based on the entered password. The service provider verifies the OTP and determines, based on the decoy seeds, whether the server has been compromised or subjected to online guessing attacks. The process of the verification phase is shown in Figure 1. The algorithms of HTOTP are as follows:

- $\{\mathbb{G}, g, k_{HC}, PK_{HC}, H(), H_{OTP}()\} \leftarrow \text{Setup}()$: The service provider executes $\text{Setup}()$ to initialize. $\text{Setup}()$ generates a private key k_{HC} for the honeychecker, and returns the public parameters $\{\mathbb{G}, g, PK_{HC}, H(), H_{OTP}()\}$, where $\mathbb{G}$ is a cyclic group, g is a generator of $\mathbb{G}$, PK_{HC} is the corresponding public key of k_{HC} , $H()$ and $H_{OTP}()$ are the hash function and OTP generation function, respectively.
- $\{r_c, T_1, EI_c, Q\} \leftarrow \text{Registration}(pw)$: The user (with the device) interacts with the server and the honeychecker to execute $\text{Registration}(pw)$. The device processes the password offline to generate salted hashes containing the real password and honeywords. The device then sends the salted hashes list Q and the index EI_c encrypted with PK_{HC} to the server. Finally, the device stores the salt r_c and the registration timestamp T_1 . The server stores the salted hash list Q and T_1 , and the honeychecker stores the index EI_c of the real password hash within Q .
- $\{0, 1, NULL\} \leftarrow \text{Verification}(pw')$: The user enters the candidate password pw' on the device and then inputs the generated OTP into the login terminal. The server interacts with the honeychecker to determine whether the OTP is correct, incorrect, or a generation result of guessed passwords or leaked OTP seeds. The server ultimately

TABLE II
COMPARISON OF SECURITY ATTRIBUTES AND APPLICATION REQUIREMENTS OF OTP SCHEMES

Scheme	Year	Ref.	Round*	S1	S2	S3	S4	S5	S6	S7	S8	C1	C2	C3	C4	C5	C6	C7	C8
DGTOTP (Cao et al.)	2024	[25]	2	×	×	✓	✓	⊗ ¹	×	×	×	✓	×	✓	×	✓	×	×	×
SOS (Peeters et al.)	2022	[27]	2	×	×	✓	✓	⊗ ¹	×	×	×	✓	×	✓	×	✓	×	×	✓
GTOTP (Yang et al.)	2021	[24]	2	×	×	✓	✓	⊗ ¹	×	×	×	✓	×	✓	×	✓	×	×	✓
OTPaas (Erdem et al.)	2019	[17]	5	✓	×	×	△ ⁶	△ ⁶	×	×	×	×	×	△ ⁶	△ ⁶	×	×	×	×
T/Key (Kogan et al.)	2017	[8]	1	×	×	✓	✓	⊗ ¹	×	×	×	✓	×	✓	×	✓	×	×	×
YubiKey (OATH-HOTP)	2011-	[32]	2	✓	×	✓	✓	⊗ ¹	⊗ ³	×	×	✓	✓	×	✓	✓	✓	✓	✓
YubiKey (OATH-TOTP)	2011-	[32]	2	✓	×	✓	✓	⊗ ¹	⊗ ³	×	×	✓	✓	×	✓	✓	✓	✓	✓
Yubico OTP	2010-	[28]	2	✓	×	×	✓	⊗ ¹	×	×	×	✓	×	×	×	×	✓	✓	×
Google Authenticator (HOTP)	2010-	[6]	1	✓	×	✓	✓	⊗ ¹	✓	×	×	✓	⊗ ²	✓	△ ⁴	×	✓	✓	✓
Google Authenticator (TOTP)	2010-	[7]	1	✓	×	✓	✓	⊗ ¹	✓	×	×	✓	⊗ ²	✓	△ ⁴	×	✓	✓	✓
Proposed HTOTP	Current	\	1	✓	✓	✓	✓	⊗ ¹	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

✓ : Achieving (Resisting) the corresponding goal (attack). × : Not achieving (Resisting) the corresponding goal (attack). \ : Not considered.
⊗ : Achieving (Resisting) with additional conditions. △ : Depending on other implementation conditions.

*: Interaction rounds during the OTP verification. ¹: Replay is valid within the time window.

²: Original HOTP [6] and TOTP [7] achieve, Google adds addition online password authentication.

³: Yubico Authenticator adds password authentication. ⁴: Depends on the password verification strategy of Google Authenticator.

⁵: Service provider is compromised, user passwords leak, and no need for OTP verification.

⁶: Depends on the OTP verification strategy adopted by third-party OTP service providers.

S1 : Resistance to device loss.

S2 : Resistance to server compromise.

S3 : Resistance to channel hijacking.

S4 : OTP unlinkability.

S5 : Resistance to replay attacks.

S6 : Two (multi)-factor security.

S7 : Secret leakage alert.

S8 : Guessing attack alert.

C1 : No verifier table.

C2 : No online interaction.

C3 : Easy to back up and restore.

C4 : Password/OTP friendly.

C5 : Resistance to keyboard/screen recording attacks.

C6 : No additional expenses.

C7 : No periodic synchronisation.

C8 : No additional third party.

receives one of three results $\{0, 1, NULL\}$ and responds accordingly with an alert, login, or re-login action.

- $El_c \leftarrow \text{Enc}_{PK_{HC}}(I_c), I_c \leftarrow \text{Dec}_{k_{HC}}(El_c)$: The user side encrypts the index I_c using the honeychecker's public key PK_{HC} , and the honeychecker decrypts the ciphertext El_c using the private key k_{HC} . The encryption and decryption algorithms are variants of ElGamal.
- $\{0, 1\}^l \leftarrow H(\{0, 1\}^*), OTP \leftarrow H_{OTP}(\{0, 1\}^*, T)$: $H(\cdot)$ is a one-way hash function that hashes an input of arbitrary length into a fixed-length output. $H_{OTP}(\cdot)$ is an OTP generation function that takes an OTP seed and a timestamp as inputs and outputs a six-digit OTP. $H_{OTP}(\cdot)$ consists of two functional components responsible for hashing and truncating, respectively.

C. Threat Model and Security Definition

In HTOTP, the adversary's goals are to obtain the user's password and forge the user's login respectively. The adversary may compromise any one of the user device, the server, or the honeychecker, but it is difficult to compromise more than one components simultaneously [44]. According to the Dolev-Yao threat model [46], the adversary may intercept and store the communication content (e.g., OTP in this scheme). The adversary may be insider in [47] the service provider and obtain the data stored by the server and the honeychecker.

The adversary's objectives are divided into guessing password and forging login. We formally define the adversary games to measure the advantage of attacking HTOTP under different capabilities. Here, "advantage" refers to the difference between the success probability and the random probability, measuring the extent to which the attack's effectiveness exceeds random guessing. When the advantage is negligible, the guessing approximates randomness.

1) Password/Login Security Under Device Compromising:

In this scenario, we assume that the adversary compromises the user's device, with the goal of guessing password and impersonating the user for login. For the sake of discussion,

we refer to the service provider composed of the server and the honeychecker collectively as the server. Here, we define two threat models for password guessing and login impersonation, which are used in Section V to demonstrate the adversary's advantage in launching above attacks after compromising device.

First, we define a game to measure the adversary's advantage in guessing password. In HTOTP, the user's device stores only the salt r_c and the timestamp T_1 after registration, without storing the hashed password. Therefore, the adversary cannot guess password offline after compromising the device. This game is denoted as PS-DC and involves two phases. In the first phase, an honest device D and an honest server S interact for registration, with D and S referred to as challenger $\mathcal{R}_D$ and $\mathcal{R}_S$, respectively. In the second phase, the server continues from the first phase, but the device becomes malicious, denoted as adversary $\mathcal{A}$. The game proceeds as follows:

- Setup: $\mathcal{R}_S$ initializes the system by executing $\Pi.\text{Setup}$, $\mathcal{R}_S$ can access the oracle $\mathcal{O} \leftarrow \{\Pi.\text{Setup}, \Pi. < \mathcal{R}_D, \mathcal{R}_S >_X, X \in \{\text{reg}, \text{verf}\}\}$. $\mathcal{A}$ generates two different passwords, pw_0 and pw_1 , and sends them to $\mathcal{R}_D$. $\mathcal{R}_D$ randomly selects one of the passwords, the corresponding password is recorded as pw_b , $b \in \{0, 1\}$. $\mathcal{R}_D$ executes $\Pi. < \mathcal{R}_D(pw_b), \mathcal{R}_S >_{\text{reg}}$.
- Challenge: The adversary $\mathcal{A}$ inherits the challenger $\mathcal{R}_D$'s parameters, and possesses the passwords pw_0 , pw_1 , the salt r_c , and the timestamp T_1 . $\mathcal{A}$ is allowed only one verification interaction to determine the guess, that is, $\mathcal{A}$'s goal is to guess the bit b selected by $\mathcal{R}_S$.
- Output: $\mathcal{A}$ selects the password $pw_{b'}$ and executes $\Pi. < \mathcal{R}_D(pw_{b'}), \mathcal{R}_S >_{\text{verf}}$. $b' = b$ represents $\mathcal{A}$ wins the game. The advantage is defined as

$$\text{Adv}_{PS-DC}^{\mathcal{A}}(\lambda) = |\Pr[b' = b] - \frac{1}{2}|. \quad (1)$$

Definition 1 (Password Security under Device Compromising): If for an OTP scheme, an adversary cannot achieve

advantage $Adv_{PS-DC}^A(\lambda)$ greater than a negligible function $negl(\lambda)$ in Probabilistic Polynomial Time (PPT), the scheme satisfies password security under device compromising.

Second, we define a game to measure the adversary's advantage in impersonating login. There are two scenarios in which the adversary can successfully log in: one is by guessing the password after compromising the device and thereby generating the correct OTP, and the other is colliding the dynamic OTPs. The adversary's advantage in achieving the former is $Adv_{PS-DC}^A(\lambda)$. Next, we define a two-phase game C-OTP to assess the advantage of colliding OTPs. In the first phase, an honest device $\mathcal{D}$ and an honest server $\mathcal{S}$ interact for registration, with $\mathcal{D}$ and $\mathcal{S}$ referred to as challenger $\mathcal{R}_D$ and $\mathcal{R}_S$, respectively. In the second phase, the server continues from the first phase, but the device becomes malicious, denoted as adversary $\mathcal{A}$. The game proceeds as follows:

- Setup: $\mathcal{R}_S$ initializes the system by executing $\Pi.\text{Setup}$, $\mathcal{R}_S$ can access the oracle $\mathbb{O} \leftarrow \{\Pi.\text{Setup}, \Pi. < \mathcal{R}_D, \mathcal{R}_S >_X, X \in \{\text{reg}, \text{verf}\}\}$. $\mathcal{R}_D$ generates a password pw , and then executes $\Pi. < \mathcal{R}_D(pw), \mathcal{R}_S >_{\text{reg}}$. $\mathcal{R}_D$ deletes pw , stores the salt r_c and the timestamp T_1 .
- Challenge: The adversary $\mathcal{A}$ inherits the challenger $\mathcal{R}_D$'s parameters, including the salt r_c and the timestamp T_1 . $\mathcal{R}_S$ generates a correct OTP OTP_b ($b \in \{0, 1\}$) based on the registration record and simultaneously generates another different random OTP OTP_{1-b} . $\mathcal{R}_S$ sends both OTPs to $\mathcal{A}$. $\mathcal{A}$ is allowed only one verification interaction to determine the guess. $\mathcal{A}$'s goal is to guess which OTP is correct.
- Output: $\mathcal{A}$ selects the OTP $OTP_{b'}$ and executes $\Pi. < \mathcal{R}_S(OTP_{b'}, OTP_b) >_{\text{verf}}$. $b' = b$ represents $\mathcal{A}$ wins the game. The advantage is defined as

$$Adv_{C-OTP}^A(\lambda) = |\Pr[b' = b] - \frac{1}{2}|. \quad (2)$$

Definition 2 (Login Security Under Device Compromising): If for an OTP scheme, an adversary cannot achieve attack advantages $Adv_{PS-DC}^A(\lambda)$ and $Adv_{C-OTP}^A(\lambda)$ greater than negligible function $negl(\lambda)$ within PPT, then we say the scheme satisfies login security under device compromising.

2) *Password Security Under Server Compromising:* When the server is compromised in the OTP authentication system, the adversary gets access to crucial verification parameters like OTP seeds and password hashes. This enables the adversary to bypass the login verification and log in as a legitimate user, making login security unachievable. So, we focus on password security instead. For the purpose of analysis, we postulate an enhanced adversary capability scenario where the adversary compromises both the server and the honeychecker maintained by the service provider. The honeychecker stores the index of the real password. By making this assumption, we can conduct a more comprehensive and stringent evaluation of the adversary's advantage in guessing the user's password after obtaining the real secret stored on the server. If the password security can be effectively ensured under such an extreme assumption, it implies that in real-world scenarios, when facing more common attack situations, the security of this OTP scheme will be more reliable, and it can better protect users' passwords and account security.

This game is defined as PS-SC and comprises two phases. In the first phase, an honest device $\mathcal{D}$ and an honest server $\mathcal{S}$ interact for registration, with $\mathcal{D}$ and $\mathcal{S}$ referred to as challenger

$\mathcal{R}_D$ and $\mathcal{R}_S$, respectively. In the second phase, $\mathcal{D}$ continues from the first phase, but the server becomes malicious, denoted as adversary $\mathcal{A}$. The game proceeds as follows:

- Setup: $\mathcal{R}_S$ initializes the system by executing $\Pi.\text{Setup}$, $\mathcal{R}_S$ can access the oracle $\mathbb{O} \leftarrow \{\Pi.\text{Setup}, \Pi. < \mathcal{R}_D, \mathcal{R}_S >_X, X \in \{\text{reg}, \text{verf}\}\}$. $\mathcal{A}$ generates two different passwords, pw_0 and pw_1 , and sends them to $\mathcal{R}_D$. $\mathcal{R}_D$ randomly selects pw_b ($b \in \{0, 1\}$) and executes $\Pi. < \mathcal{R}_D(pw_b), \mathcal{R}_S >_{\text{reg}}$.
- Challenge: The adversary $\mathcal{A}$ inherits the challenger $\mathcal{R}_S$'s parameters, including the salted hash $q_c \leftarrow H(pw_b \parallel r_c)$ and the timestamp T_1 , where r_c is the salt stored by $\mathcal{R}_D$. $\mathcal{A}$'s goal is to guess the bit b selected by $\mathcal{R}_D$. $\mathcal{A}$ can access the oracle $\mathbb{O} \leftarrow \{\Pi.\text{Setup}, \Pi. < \mathcal{R}_D, \mathcal{R}_S >_{\text{verf}}\}$.
- Output: $\mathcal{A}$ determines the password $pw_{b'}$. $b' = b$ represents $\mathcal{A}$ wins the game. The advantage is defined as

$$Adv_{PS-SC}^A(\lambda) = |\Pr[b' = b] - \frac{1}{2}|. \quad (3)$$

Definition 3 (Password Security Under Server Compromising): If for an OTP scheme, an adversary cannot achieve an attack advantage $Adv_{PS-SC}^A(\lambda)$ greater than a negligible function $negl(\lambda)$ within PPT, then we say the scheme satisfies password security under server compromising.

IV. THE PROPOSED HTOTP

In this Section, we first discuss the construction idea of the tightly coupled two-factor OTP and describe in detail the specific steps of the scheme.

A. Construction Overview

In addition to password authentication, OTP serves as the second factor, representing the device in the authentication process. Device loss or OTP secret leakage pose significant threats to the security of authentication factors. Therefore, it is crucial to strengthen the security of these factors. Additionally, in the event of a leak, it is essential to resist offline guessing attacks and potential threats to password security.

The design goals and construction ideas are as follows:

1) *Unforgeability of the Second Factor:* Devices are utilized as the second factor, typically generating [7], [18] or receiving OTPs [12] in the 2FA architecture. However, we observe that in the specific implementation of Google 2FA (device and password) [9], OTPs don't necessarily equate to device possession, and adversaries can steal OTPs through various means (such as SMS interception, screen capture to obtain OTP generation seeds, server/device compromise), thereby bypassing the device. Our objective is to establish a robust correlation between OTPs and the user's possession of the device. The correlation ensures that even if adversaries acquire the device, they cannot impersonate the user to generate OTPs.

2) *Tightly Coupled Two Factors:* The loosely coupled 2FA, such as Google 2FA, has independent password and device authentication. The main challenge is that when adversaries compromise the server, they can simultaneously breach both the password and OTP seed. More generally, adversaries can compromise one factor while the other factor fails to provide the necessary obstruction. Our goal is to tightly couple two factors of password and device, making it difficult for adversaries to bypass one factor to attack the other. An additional objective is to make it difficult for adversaries to compromise the other factor even if they have already obtained one factor.

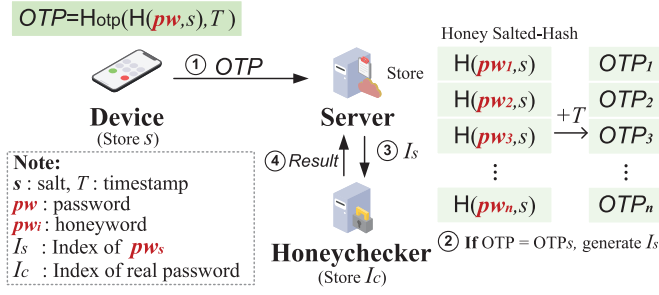


Fig. 1. Overview of OTP verification. The user device first calculates OTP, where $OTP = H_{otp}(H(pw, s), T)$, with $H_{otp}()$ and $H()$ are hash functions. After receiving the OTP, the server generates OTPs using the stored salted honeyword hash value and compares them with the received OTP. The index I_s obtained from the comparison is sent to the honeychecker. The honeychecker compares I_s with the real password index I_c and returns the verification result to the server.

3) *Resisting Offline Password Guessing Attacks/Key Compromise Impersonation (KCI) Attacks*: In the existing implementation of 2FA that relies on passwords and OTPs, adversaries can steal OTP seeds and launch offline password guessing attacks after compromising server data, thereby initiating KCI attacks. Our goal is to prevent adversaries from guessing user passwords or directly obtaining OTP seeds after compromising the server.

4) *Secret Leakage/Online Password Guessing Alert*: Server data leakage and user device loss are typically difficult for servers to detect and pose significant threats to user authentication. Our goal is for the server to identify and alert when there is a server data breach or offline guessing based on devices.

To achieve Goals 1 and 2, it is essential to ensure the security of the OTP seed shared between the device and the server, thereby preventing adversaries from obtaining the seed. We enhance security by shifting the secret-sharing approach from server-driven distribution to device-driven distribution. The OTP seed consists of a user password and the salt value generated by the device. After registration, the device stores a salt value, ensuring that only the user with the device can generate correct OTPs. Moreover, adversaries cannot verify the other factor offline after obtaining one factor.

Furthermore, to prevent offline password guessing as outlined in Goal 3, the server and the user device store the password salted hash and the salt separately. Additionally, the server stores multiple honey password salted hashes for each account simultaneously. Consequently, even if the adversary compromises the server, they cannot distinguish the real OTP seed from other decoy seeds, and cannot verify the correctness of the guessed passwords without knowing the salt.

To achieve Goal 4, Honeywords incorporates popular passwords and decoy passwords derived from user personal information. When an adversary generates an OTP based on decoy OTP seeds after compromising the server or generates OTPs based on guessed passwords after compromising the device, the honeychecker can compare these OTPs to detect data leakage or password guessing, thereby triggering an alert.

Next, we introduce the specific scheme steps.

B. Scheme Description

1) *Setup Phase*: The service provider selects a cyclic group $\mathbb{G}$ of order q with a generator g . $H : \{0, 1\}^* \rightarrow \{0, 1\}^l$ represents a one-way hash function. $H_{OTP}(\{0, 1\}^l, \{0, 1\}^{32}) =$

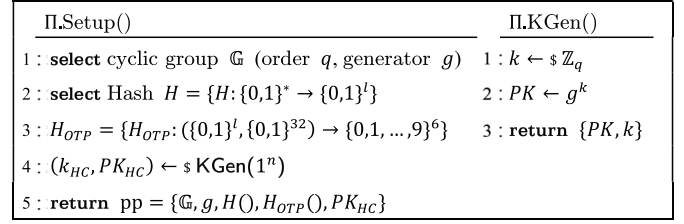


Fig. 2. The main operations of setup phase.

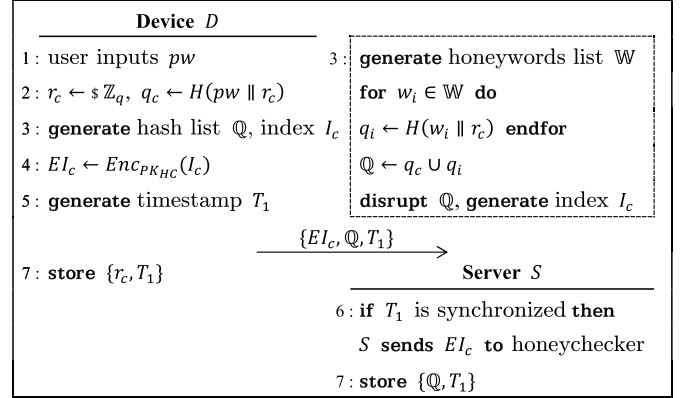


Fig. 3. The main operations of registration phase.

$Truncate(HMAC - SHA - 1(\{0, 1\}^l, \{0, 1\}^{32}))$ represents the OTP generation function [7], where $HMAC - SHA - 1 : (\{0, 1\}^l, \{0, 1\}^{32}) \rightarrow \{0, 1\}^{160}$ is a one-way hash function [18], and $Truncate$ is a truncation function that reduces the hash value to a 6-digit number. Honeychecker generates a public-private key pair, denoted as (PK_{HC}, k_{HC}) . The authentication server and honeychecker server communicate over a secure channel. The public parameters $pp = \{\mathbb{G}, g, H(), H_{OTP}(), PK_{HC}\}$ are embedded in the authentication application downloaded by users.

The setup phase is illustrated in Figure 2.

2) *Registration Phase*: The verification phase is illustrated in Figure 3, with the corresponding description as follows:

R1: The user U inputs a password pw on the device D . Afterwards, D randomly selects a salt value r_c from $\mathbb{Z}_q$ and computes the salted hash $q_c \leftarrow H(pw \parallel r_c)$ of the password. D generates a honeywords list $\mathbb{W}$, for each honeyword w_i in the list, D computes the corresponding salted hash $q_i \leftarrow H(w_i \parallel r_c)$. The list composed of the salted hash q_c of the real password and the salted hashes q_i of honeywords is referred to as sweet-hashes list $\mathbb{Q} \leftarrow q_c \cup q_i$. D disrupts $\mathbb{Q}$ and generates an index I_c to indicate q_c . Then, D encrypts I_c to $El_c \leftarrow Enc_{PK_{HC}}(I_c)$ using honeychecker's public key PK_{HC} , the encryption is based on a variant of ElGamal. During encryption, D selects $e \leftarrow \mathcal{Z}_q$, and calculates $E \leftarrow g^e$ and $M \leftarrow PK_{HC}^e \oplus I_c$, $El_c = \{E, M\}$, where $\oplus$ is XOR operation. Finally, D generates these current Unix timestamp T_1 and sends $\{El_c, \mathbb{Q}, T_1\}$ to the server S through a secure channel. Note that the device can conveniently initiate a secure channel establishment using the server's public key. D stores $\{r_c, T_1\}$.

R2: On receiving $\{El_c, \mathbb{Q}, T_1\}$, the server S first verifies the freshness of T_1 . Specifically, S generates the current timestamp T'_1 and verifies if $|T'_1 - T_1| < \Delta T$, where ΔT is the maximum delay. If the clocks of S and D are synchronized, S sends El_c to the honeychecker C and stores $\{\mathbb{Q}, T_1\}$.

Device D	Server S
1: user inputs pw' on device	
2: $q'_c \leftarrow H(pw' \parallel r_c)$	
3: generate timestamp T_2	7: receive OTP
4: $T_S \leftarrow (T_2 - T_1)/X$	8: generate timestamp T_3
5: $OTP \leftarrow H_{OTP}(q'_c, T_S)$	9: $T'_S \leftarrow (T_3 - T_1)/X$
6: user inputs OTP	10: for $q_i \in \mathbb{Q}$ do $OTP_i = H_{OTP}(q_i, T_S)$ endfor
Honeychecker C	
13: C return 0, alert	11: if $OTP \neq OTP_{x(q_x \in \mathbb{Q})}$ abort
C return 1, login	12: if $OTP = OTP_x = OTP_y (x \neq y)$ or $OTP = OTP_x \neq OTP_y (x \neq y)$ send index(s) to honeychecker
C return NULL, re-login	

Fig. 4. The main operations of verification phase.

R3: On receiving EL_c , the honeychecker C decrypts EL_c by calculating $I_c = Dec_{k_{HC}}(EL_c) = E^{k_{HC}} \oplus M$.

The server and the honeychecker are independent entities that interact through a secure channel. They store the honey list and the index of the real secrets, respectively. Compromising both the server and the honeychecker is difficult for adversaries. Furthermore, even if an adversary compromises the real salted hash $q_c = H(pw \parallel r_c)$, they cannot guess the password offline without knowing the salt r_c .

The registration process of HTOTP differs from Google Authenticator in that the shared keys are distributed from the device to the server rather than the server distributing keys. In the case of Google Authenticator, the server generates a shared key and converts it into a QR code, which users scan to obtain the key. On the one hand, upon decoding, we discover that the QR code contains the username and secret in plaintext. Since the QR code remains static, any adversary can capture the user's identity and long-term secret through methods like screen capture or shoulder surfing attacks. On the other hand, strengthening the method of secret sharing doesn't appear to be the optimal solution. If the server initiates to establish a secure channel, it would require the device to maintain and share a public key, incurring additional storage and computational overhead for both the device and the server. More importantly, secret generation and distribution by the server always remain vulnerable to attacks caused by server breaches.

3) *Verification Phase:* The verification phase is illustrated in Figure 4, with the corresponding description as follows:

V1: The user U first enters a candidate password pw' into the device D , then D computes the corresponding salted hash $q'_c \leftarrow H(pw' \parallel r_c)$. Note that D does not store password verification information, so even if an adversary obtains D , she cannot use it for offline password guessing. Afterward, D generates the current Unix timestamp T_2 and calculates the time difference $T_S \leftarrow (T_2 - T_1)/X$ with the registration timestamp T_1 , where X represents the time step, typically set to 30 seconds, meaning that within a 30-second window, both parties generate the same OTP based on the same secret. D calculates $OTP \leftarrow H_{OTP}(q'_c, T_S)$ using the time difference T_S and the salted hash q'_c , where $H_{OTP}(q'_c, T_S) = Truncate(HMAC - SHA - 1(q'_c, T_S))$, $Truncate$ represents the truncation function [7], truncating the hash value $HMAC - SHA - 1(q'_c, T_S)$ to a 6-digit number. U inputs the OTP on terminal.

V2: On receiving OTP, the server S generates the current Unix timestamp T_3 and calculates the time difference $T'_S \leftarrow (T_3 - T_1)/X$. Then, S calculates the corresponding OTPs for hashes in the sweetshashes list $\mathbb{Q}$. If the received OTP does not match any of the calculated candidate OTPs, the user's login attempt fails, and the server terminates the process. If the received OTP matches one or more candidate OTPs, the server sends the index(s) of the matching OTP(s) to the honeychecker C via a secure channel.

V3: On receiving index(s), the honeychecker C first verifies whether the received index(s) match(es) the correct index I_c . If there is no match, honeychecker returns 0, indicating a warning that the server may have leaked data or that an adversary is guessing the password on the device. If the single index matches the correct index, honeychecker returns 1, indicating successful login. If there is a match among multiple indexes, indicating an OTP collision, honeychecker returns a blank message, prompting the server to re-authenticate the user.

Please note that the server first generates the current timestamp T_3 upon receiving the message and then proceeds with OTP calculation and comparison. The time difference between T_3 and T_2 is primarily due to transmission delay. The server's computation and comparison overhead do not cause delays that would affect the accuracy of OTP verification.

During the verification phase of HTOTP, the user inputs the OTP generated offline into the login terminal. The usage of OTP aligns with the user's accustomed usage of Google Authenticator. We modify the underlying logic of Google's 2FA, transitioning from loosely coupled to tightly coupled factors. Two factors refer to the password and the device. The server authenticates the device by verifying the OTP derived from the secret stored on the device. Loosely coupled 2FA means that password authentication and OTP authentication are independent, with Google Authenticator separately (step-by-step) authenticating the password and OTP. Therefore, it is easier for adversaries to compromise individual factors separately than to compromise both factors simultaneously.

In HTOTP, the device integrates stored salt and passwords to derive OTPs, transforming the OTP from solely representing the device to simultaneously representing both the password and the device. It is challenging for adversaries to compromise the tightly coupled password-device combination. Moreover, the introduction of honeywords prevents adversaries compromising the server from discerning the real secret and guessing the password without knowing the salt. Adversaries who attempt to generate OTP directly after compromising the server, or who guess the password to generate OTP after obtaining the device (salt), may trigger alerts from the honeychecker, leading to an account freeze. In terms of practicality, users can back up the salt on other clock-synchronized devices to enhance availability.

4) *Password Change Phase:* In the event of device loss or a honeychecker alert, account security can be restored by rotating the passwords. The steps are as follows:

PC1: The user U inputs the previous password pw and the new password pw^n on the device D . D selects a random salt r_c^n from $\mathbb{Z}_q$ and computes the salted hash $q_c^n \leftarrow H(pw^n \parallel r_c^n)$. D generates a new honeywords list $\mathbb{W}^n$, for $w_i^n \in \mathbb{W}^n$, D computes $q_i^n \leftarrow H(w_i^n \parallel r_c^n)$, $\mathbb{Q}^n \leftarrow q_c^n \cup q_i^n$. D disrupts $\mathbb{Q}^n$ and generates an index I_c^n to indicate q_c^n . Then, D encrypts I_c^n

to $El_c^n \leftarrow \text{Enc}_{PK_{HC}}(I_c^n)$ using honeychecker's public key PK_{HC} . D computes $q_c \leftarrow H(pw \parallel r_c)$. Finally, D generates the current Unix timestamp T_n and sends $\{q_c, El_c^n, Q^n, T_n\}$ to the server S through a secure channel.

PC2: S sends q_c, El_c^n, Q to the honeychecker. If the verification of q_c is successful, S replaces Q with Q^n , and the honeychecker replaces El_c with El_c^n . D replaces r_c with r_c^n .

V. SECURITY ANALYSIS

In this section, we formally prove the adversary's advantage in launching attacks when compromising the user device or the server, and heuristically discuss the probability of the adversary's successful attacks.

A. Formal Security Proof

The hash functions in this work are assumed to possess the properties of pre-image resistance, second-pre-image resistance, and collision resistance. Pre-image resistance means that given a hash value $h = H(x)$, it is computationally infeasible to find an input x such that $H(x) = h$. Second-pre-image resistance implies that given an input x_1 , it is computationally infeasible to find a different input $x_2 \neq x_1$ such that $H(x_1) = H(x_2)$. Collision resistance ensures that it is computationally infeasible to find two different inputs $x_1 \neq x_2$ such that $H(x_1) = H(x_2)$.

1) Password Security Under Device Compromising:

Theorem 1: Let Adv_{TLS}^A represents the adversary's advantage in breaking TLS to obtain the content, and Adv_{CS}^A represents the adversary's advantage in compromising the server to obtain the stored content. then the adversary's advantage in determining the password after compromising the device can be represented as:

$$\text{Adv}_{PS-DC}^A(\lambda) \leq \max(\text{Adv}_{TLS}^A, \text{Adv}_{CS}^A) \quad (4)$$

Proof: The validity of this theorem is established through the formalization of the game, where the advantage lies in identifying the bit $b \in \{0, 1\}$, representing the adversary's determination of which of the two passwords is the registered one. When the adversary's advantage is negligible, she has no significant advantage in determining the password.

To break down the proof, we analyze the adversary's actions through a series of games. Each game incrementally reveals more information to the adversary, allowing us to measure their advantage in different scenarios.

- **Game₀:** This game is identical to the game PS-DC described in Section III-C. The adversary's advantage is quantified as the absolute difference between the probability that the adversary correctly identifies the bit b . The adversary's goal is to determine which of the two passwords (pw_0 or pw_1) is the registered one. Formally, we have:

$$\text{Adv}_{PS-DC}^A(\lambda) = |\Pr[b' = b] - \frac{1}{2}|.$$

- **Game₁:** In this game, the adversary $\mathcal{A}$ has access to the passwords pw_0 , pw_1 and the salt r_c , but not the salted hash $q_c \leftarrow H(pw_b \parallel r_c)$ stored on the server. Since the adversary lacks the salted hash, they cannot directly verify which password is correct. The adversary's probability of correctly determining b is $\frac{1}{2}$, implying that:

$$\text{Adv}_{PS-DC}^A(\lambda) = |\Pr[b' = b] - \frac{1}{2}| \leq \text{negl}(\lambda).$$

- **Game₂:** Here, we consider the adversary having access to the compromised device's state, including nonces or intermediate values used during the OTP generation process. However, without the salted hash q_c , the adversary still cannot effectively distinguish between the two passwords. Unless the adversary intercepts the message during registration and has the ability to break the TLS to obtain the message. Otherwise, the security properties of the hash function H ensure that any attempt to guess the password is no better than random guessing. Consequently, the probability is:

$$\text{Adv}_{PS-DC}^A(\lambda) \leq \text{Adv}_{TLS}^A.$$

- **Game₃:** In this final game, we account for any additional auxiliary information that the adversary might obtain from the compromised server, such as partial state leakage, the advantage is Adv_{CS}^A . Thus, the adversary's advantage can be expressed as:

$$\text{Adv}_{PS-DC}^A(\lambda) \leq \max(\text{Adv}_{TLS}^A, \text{Adv}_{CS}^A).$$

By analyzing the adversary's advantage through these games, we demonstrate that even with access to significant amounts of information from the compromised device, the adversary cannot gain a significant advantage in determining the user's password. This proof establishes that the adversary's advantage in determining the user's password after compromising the device is bounded by a negligible function, ensuring robust security.

2) Login Security Under Device Compromising:

Theorem 2: The adversary's advantage in distinguishing OTPs can be represented as:

$$\text{Adv}_{C-OTP}^A(\lambda) = \text{Adv}_{PS-DC}^A(\lambda) \quad (5)$$

Proof: The adversary's advantage in distinguishing OTPs when compromising the device is the same as the advantage in distinguishing passwords. The proof is similar to *Proof 1*.

3) Password Security Under Server Compromising:

Theorem 3: Let $\text{Adv}_{Hash}^A(\lambda)$ represent the adversary's advantage in reversing the hash to obtain the original value, $\text{negl}(\lambda)$ represent a negligible function. Then, the probability of the adversary obtaining the password after compromising the server can be represented as:

$$\text{Adv}_{PS-SC}^A(\lambda) \leq \text{Adv}_{Hash}^A \quad (6)$$

Proof: The theorem's validity is shown through the formalization of the game, where the advantage lies in identifying the bit $b \in \{0, 1\}$, representing the adversary's determination of which password is correct after compromising the server. When the adversary's advantage is negligible, she has no significant advantage in guessing passwords.

To prove it thoroughly, we consider a series of hypothetical games, each building on the previous one to establish the password's security under the condition of server compromise.

- **Game₀:** This game is identical to the game PS-SC as described in Section III-C. In this scenario, the adversary's advantage is measured by their ability to correctly identify the bit b , which indicates whether a given password guess is correct. The adversary's advantage is defined as:

$$\text{Adv}_{PS-SC}^A(\lambda) = |\Pr[b' = b] - \frac{1}{2}|.$$

- **Game₁**: In this game, the adversary $\mathcal{A}$ makes a direct guess at the bit $b \in \{0, 1\}$. Without any additional information, the best strategy for the adversary is to guess randomly. Therefore, the probability of the adversary correctly determining b is $\frac{1}{2}$, resulting in:

$$|\Pr[b' = b] - \frac{1}{2}| \leq \text{negl}(\lambda).$$

- **Game₂**: In this game, the adversary $\mathcal{A}$ has access to both candidate passwords pw_0 and pw_1 , the salted hash $q_c \leftarrow H(pw_b \parallel r_c)$, and the timestamp T_1 . However, she do not have the salt r_c . Without the salt, the adversary cannot generate the correct salted hash to compare against q_c . Unless the adversary can recover the password pw_b from the hash q_c , its advantage in this game remains the same as in **Game₁**. The adversary's ability to reverse the hash to obtain the password is denoted as $\text{Adv}_{\text{Hash}}^{\mathcal{A}}$.

Combining the results of these games, we can conclude:

$$\text{Adv}_{\text{PS-SC}}^{\mathcal{A}}(\lambda) \leq \text{Adv}_{\text{Hash}}^{\mathcal{A}}.$$

This means that the adversary's advantage in guessing the password after compromising the server is at most as large as their advantage in reversing the hash function. Since modern hash functions resist inversion, recovering the password from the hash is negligible, thereby ensuring the security of the password under server compromise.

In conclusion, by formalizing the adversary's advantage through these games, we demonstrate that the probability of an adversary successfully guessing the password after a server compromise is effectively bounded by their ability to reverse the hash function. This strongly ensures password security, provided the underlying hash function resists inversion attacks.

B. Further Security Discussion

In this subsection, we heuristically analyze the satisfaction of HTOTP for the security criteria involved in Section III-A, and informally discuss the resistance of the scheme against real-world attacks.

1) *Resistance to Device Loss*: The device stores the salt of the password hash. An adversary compromising the device cannot directly obtain the other authentication factor (password) or generate the correct OTPs for login unless online guessing is performed. According to the research of Wang et al. [2], the password set conforms to the Zipf distribution. The probability of correctly guessing the password after obtaining the device is no more than $C' \cdot q_{\text{guess}}^{s'}$, where q_{guess} is the number of guesses, C' and s' are the linear regression parameters of the password set $\mathbb{D}$ that conform to Zipf's law. The probability of the adversary guessing the OTP correctly is $q_{\text{guess}}/2^{l_{\text{OTP}}}$, where l_{OTP} is the bit length of OTP. To summarize, the probability that the adversary forges the user's login with q_{guess} times online guesses after obtaining the device is $\max(q_{\text{guess}}/2^{l_{\text{OTP}}}, C' \cdot q_{\text{guess}}^{s'})$, and the probability of breaking the user's password is at most $C' \cdot q_{\text{guess}}^{s'}$.

2) *Resistance to Server Compromise (and Insider Attack)*: The server stores the salted hashes of the passwords, while the corresponding salt is retained on the device. Even if an

adversary manages to compromise the server's data (and this adversary can also be regarded as an insider attacker), they are unable to conduct offline password guessing verification without knowing the salt. In addition to the salted hash of the real password, the server also stores the salted hashes of $k - 1$ honeywords. Unless the original content of the hashes is recovered, the adversary cannot distinguish the salted hashes according to the semantics. The probability that the adversary distinguishes the real OTP seed is $1/k$. That is to say, the adversary's login attempt has a probability of $(k - 1)/k$ of being recognized by the honeychecker, which issues a warning.

Assume that the adversary is an insider attacker within Honeychecker. They can obtain the index of the correct password among the honeywords, but they are unable to learn the user's password or the salted hash corresponding to the index. Therefore, the insider attacker within Honeychecker cannot launch subsequent attacks.

Assume that the adversary is an insider attacker within the service provider and can compromise both the server and Honeychecker. In this scenario, it is simplistic for the adversary to forge user logins. At this point, the adversary has already obtained the salted hash of the user's actual password, and their objective is to crack the user's password. However, since the salt is stored on the user's device, the adversary is unable to verify their password guesses. In summary, the above analysis clearly demonstrates that our proposed scheme is effective in withstanding insider attacks.

3) *Resistance to Channel Hijacking*: In general, the registration stage is initiated by the device leveraging the service provider's public key, and the established TLS channel ensures the secrecy of the transmitted messages. Without the salt used in the salted hash mechanism, even if the adversary hijacks the registration channel, she cannot break the user's password. In addition, at the verification stage, both OTP generation by the device and user OTP input occur offline, rendering channel hijacking ineffective.

4) *OTP Unlinkability and Resistance to Replay Attacks*: A one-way hash function has an avalanche effect, and a slight difference in input will lead to completely different results. Unless the adversary can restore the input of the hash function from the truncated hash value, it is impossible to judge whether there is a correlation between OTPs. In addition, time-based OTP is only valid within a time window. The basis for replaying an expired OTP to be valid is that the replayed OTP is the same as the current OTP, with a probability of $1/2^{l_{\text{OTP}}}$, where l_{OTP} is the bit length of OTP.

5) *Two (Multi)-Factor Security*: The authentication factors consist of the password and the device. The adversary cannot forge the user login if she obtains only one of factors. When an adversary possesses the user's device, she must attempt online password guessing via login attempts, and such actions can be detected and trigger the guessing alert. When the adversary obtains the user password, she lacks the device (where the salt is stored), and thus cannot calculate OTP seed to generate correct OTPs. Therefore, the proposed HTOTP has two-factor security.

6) *Secret Leak and Guessing Attack Alert*: The server stores the list of honey hashes composed of decoy passwords.

TABLE III
OVERHEAD OF BASIC OPERATIONS

Operation	Server Overhead ¹ (μs)	Phone Overhead ² (μs)
Hash(SHA-256)	0.9961-2.173	25.14-1694
Hash(HMAC-SHA-1)	2.812-7.454	63.85-4823
Enc/Dec(AES-128)	203.7-524.9	974.46-25913
ECC point mul(P-256)	34.61-105.48	1672-48135

1: Server performance measurements were conducted on the following devices:
 Ubuntu (22.04.3 LTS), Intel Xeon Gold 6320R@2.10GHz, 256GB of RAM, NVIDIA RTX 3090 GPU.
 Ubuntu (22.04.3 LTS), Intel Core i7-12700H@2.30GHz, 16GB of RAM, NVIDIA RTX 3060 GPU.
 Ubuntu (20.04.3 LTS), Intel Core i5-1135G7@2.40GHz, 16GB of RAM, Intel Iris Xe Graphics.
 Ubuntu (16.04 LTS), Intel Core i3-2310M@2.10GHz, 4GB of RAM, AMD Radeon HD 6370M.
 2: Phone performance measurements were conducted on the following devices:
 Android 14, MediaTek Dimensity9300+@4x3.0GHz, 12GB of RAM, Immortalis-G720 MP10 GPU.
 Android 14, Kirin9100@4x3.0GHz+4x2.5GHz, 16GB of RAM, Mali-G78 MP20 GPU.
 Android 13, MediaTek Dimensity8100@4x2.85GHz, 8GB of RAM, Mali-G610 MC6 GPU.
 Android 13, Qualcomm Snapdragon 4Gen2@2x2.2GHz+6x1.8GHz, 8GB of RAM, Adreno 705 GPU.

When the adversary employs leaked secrets or guessed popular passwords for login attempts, the honeychecker can detect decoys in the list and activate alerts.

VI. EVALUATION

We now evaluate the efficiency of the proposed HTOTP, and make comparisons among related schemes in terms of overhead and usability.

Deployment of the Experimental Environment: The server is built on Ubuntu (Version: 22.04.3 LTS), and the hardware specification is Intel Xeon Gold 6320R @ 2.10GHz, 256GB of RAM, NVIDIA RTX 3090 GPU. In the server program, we use the Mac class under the javax.crypto package to implement the HMAC-SHA-1 algorithm, and the construction of the Truncation function refers to RFC4226 [6]. The login interface is a web page written in Python that connects to the server. The deployment of honeychecker is the same as that of the server.

The device program is written in JAVA and deployed on Android devices with the specification (Version: Android 13, Hardware: MediaTek Dimensity 8100 @ 4×2.85 GHz, 8GB of RAM, Mali-G610 MC6 GPU). The algorithms such as key generation, hash, and encryption involved in the setup and registration stages are provided by the JCA (Java Cryptography Architecture) library. The channel between the device and the server, and the channel between the login terminal and the server, are based on the TLS protocol under the javax.net package.

In Table III, we evaluate the overhead of basic operations across devices with varying performance levels. While the overheads measured on different platforms and hardware vary, we believe that the comparisons between schemes under the same conditions still offer valuable insights, and the variation in overhead does not significantly affect the comparative analysis of the schemes' performance.

In the verification protocol, the server immediately obtains the current timestamp after receiving the OTP, so the delay in generating the OTP between the device and the server does not include the operations involved in verifying the OTP. We set the time step X to 30 seconds, which is the same as the step of common OTP authentications such as Google Authenticator and Microsoft Authenticator. The time step aligns with user habits and does not affect the implementation of the scheme. In the given hardware environment, we measure

the communication latency between the terminal and the server to be approximately 1 ms in a Local Area Network (LAN) and 90 ms in a Wide Area Network (WAN). OTP schemes' communication latency mainly originates from the aforementioned interaction, specifically the transmission of OTP. Since the process of generating the OTP on the user device is offline and only one round of communication contributes the overhead latency, the evaluation primarily focuses on the computational overhead of the scheme. In practical applications, the time step X can be appropriately adjusted according to the communication latency and the length of the OTP to provide redundancy.

A. Performance of HTOTP

1) *Setup Phase:* The cost in the setup phase mainly lies in the generation and distribution of key pairs. We select public parameters based on the common elliptic curve P-256. The lengths of the public and private key of honeychecker are both 256 bits. The hash function is selected as SHA-256, and the output is 256 bits. We truncate HMAC-SHA-1 to a 6-digit OTP, with an entropy of approximately 19.93 bits. At this point, the entropy is approximately equal to that of the password chosen by humans (according to the study by Wang et al. [2], [48], the entropy of the human chosen passwords is 20-22 bits). If the OTP specification is set to an 8-digit number, then the entropy value is 26.58 bits, which is higher than the human chosen passwords. Consequently, users do not need to worry about the OTP being more vulnerable to collision attacks by adversaries.

2) *Registration Phase:* The registration phase involves the interaction between the user's device and the server. The device establishes a TLS channel based on the server's public key and TLS is considered secure. Here, the server's public-private key pair is only used for establishing the channel, so we omit its detailed elaboration. The device generates honeywords according to the user's password. The number of honeywords follows the recommended $k = 20$ [3].

Taking the 32M Rockyou password data set as an example, the training is based on the hybrid model of $\frac{1}{3}$ TarList + $\frac{1}{3}$ TarMarkov + $\frac{1}{3}$ TarPCFG, and a common computer only needs 1.17 ms to generate 20 honeywords. The total number of generated honeywords and popular passwords is 20. The device calculates 20 hashes and an asymmetric encryption of the index. The time cost of single hash (SHA-256) and an asymmetric encryption based on ECC (P-256) is approximately 0.9961 μs (microseconds, $1 \mu s = 10^{-6} s$) and 71.87 μs respectively. The computational cost of the device is approximately 1.121 ms. The device constructs a TLS channel using the server's public key to send messages, and the total cost of the registration stage is 1.310 ms.

3) *Verification Phase:* In the verification stage, the device generates OTP offline based on the password without interacting with the server. The device computes hash and HMAC-SHA-1 and truncates them into OTP. The time cost of a single HMAC-SHA-1 is approximately 2.999 μs . After the user inputs the password, the total time cost for the device to generate the OTP is approximately 4.1524 μs . On the server side, the server generates OTPs for 20 candidate OTP seeds and verifies them with the honeychecker. The

TABLE IV
COMPARISON OF OVERHEADS

Scheme	Registration			Verification			Storage	
	Round	Device	Server	Round	Device	Server	Device	Server
HOTP [6]	1	$H_{OTP}(67.14\mu s)$	$H_{OTP}(3.027\mu s)$	1	$H_{OTP}(67.14\mu s)$	$H_{OTP}(3.027\mu s)$	$L_{seed} + L_{C/T}$	$L_{seed} + L_{C/T}$
TOTP [7]	1	$H_{OTP}(67.14\mu s)$	$H_{OTP}(3.027\mu s)$	1	$H_{OTP}(67.14\mu s)$	$H_{OTP}(3.027\mu s)$	$L_{seed} + L_{C/T}$	$L_{seed} + L_{C/T}$
T/Key [8]	1	$kH_Z(0.9961k\mu s)$	$\backslash$	1	$t_1H_Z(0.9961t_1\mu s)$	$t_2H_Z(0.6283t_2\mu s)$	$L_{ID} + L_{HZ} + L_{C/T}$	$L_{ID} + L_{HZ} + L_{C/T}$
OTPaas [17]	7	$\backslash$	$\backslash$	5	$\backslash$	$\backslash$	$2L_{ID}$	$6L_{ID}$
GTOTP [24]	2	$2H_Z + 2E_M(78.31\mu s)$	$Mt + d(Enc + 2H_Z + 2E_M)$ (299.64d+3.027 μs)	2	$\backslash$	$Mt + H_Z + Dec$ (6.054d+215.4 μs)	$L_{En} + 2H_Z$	L_{Mt}
DGTOTP [25]	2	$8H_Z + Enc(223.4\mu s)$	$Mt + 7dH_Z(8.965d-0.9961\mu s)$	2	$\backslash$	$Mt + 4H_Z + Dec$ (5.977d+214.4 μs)	L_{En}	$L_{Mt} + 2dL_{HZ}$
HTOTP	1	$wH_Z + 2EM$ (691.88+0.9961w μs)	$2E_M(71.88\mu s)$	1	$H_Z + H_{OTP}(114.1\mu s)$	$wH_{OTP}(3.027w\mu s)$	$L_H + L_{C/T}$	$wL_H + L_{C/T}$

H_{OTP} : Hash from $\{0, 1\}^*$ to OTP.
 E_M : ECC point multiplication in P-256.
 Enc : Symmetric encryption.
 L_{En} : Length of encryption block.
 $L_{C/T}$: Length of counter/timestamp (64bits).
 w : The size of honeywords.

L_{seed} : Length of OTP seed (256bits).
 L_{ID} : Length of identity (32bits).
 Dec : Symmetric decryption.
 k : The number of nodes in the hash chain.
 $\backslash$: Not considered.
 $t_1 + t_2$: Total nodes minus previous verification nodes.

H_Z : Hash from $\{0, 1\}^*$ to $\{0, 1\}^l$.
 L_{HZ} : Length of the hash H_Z .
 Mt : Merkle tree generation.
 d : The number of user devices.
 L_{Mt} : Length of Merkle tree.

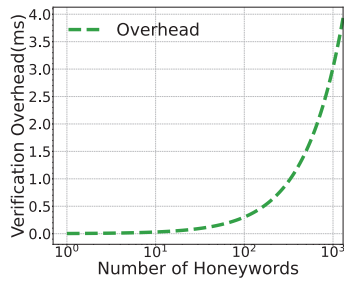


Fig. 5. The relationship between server overhead and the number of honeywords (X-axis shows exponential growth). When the number of honeywords is no greater than 100, the overhead is less than 0.3 ms.

relationship between the number of candidate OTPs and server overhead is shown in Figure 5. The time cost from receiving OTP to verifying OTP on the server is approximately stored by the device and the server. 0.2376 ms.

B. Comparison

In this subsection, we compare HTOTP with OTP implementations and the popular schemes in terms of performance and cost, and the comparison is shown in Table IV.

In the implementations of HOTP [6] and TOTP [7], we take the default registration steps of Google Authenticator and Microsoft Authenticator as examples. The device scans the QR code generated by the server to obtain the OTP seed, and both the device and the server generate an OTP for the successful verification of synchronization. In verification, the device generates an OTP based on the stored OTP seed combined with the counter (timestamp), and the cost is the same as that of registration. In addition, the OTP seeds and the counters (timestamps) are symmetrically stored by the device and the server.

In scheme T/Key [8], the device generates a hash chain of k nodes (k is 2×10^6 in [8]), while each node consists of a timestamp and a salt, and the server stores the tail node of the chain. In the verification stage, the device performs t_1 hashes according to the difference between the current time and the initialization time, and the server performs t_2 hashes according to the difference between the tail node and

the current time. Both the device and the server store the user identity, the initial (tail) node hash, and the timestamp. During the initial verification, $t_1 + t_2 = k$. Subsequently, $t_1 + t_2$ is k minus the time point indicated by the previous verification.

The scheme OTPaas [17] introduces an additional OTP provider beyond the user's device and the server, thus requiring additional interaction rounds during registration and verification. Except for the interaction framework, OTPaas does not provide a specific implementation algorithm, so it is unfair to discuss the computational overhead in table IV. In terms of storage overhead, the device stores the user name and the information of the OTP provider. The OTP provider keeps a verification table with three-party info (username, OTP seed, service provider). The service provider stores details like username, password, and the OTP provider. Here, we combine and count their storage.

In the scheme GTOTP [24], during the registration stage, the server constructs a Merkle tree according to the device nodes and encrypts the user information as the verification token, and the relationship between the generation cost of the Merkle tree and the number of devices d is $(2d - 1)H_Z$. We show that the time for the AES-128 algorithm to symmetrically encrypt a block is 215.4 μs . In addition, for fairness, here we equivalently replace the modular exponentiation multiplication of all schemes with ECC point multiplication based on P-256, a single ECC point multiplication takes 35.94 μs . At the verification stage, the server decrypts the token submitted by the user to generate the OTP.

DGTOTP [25] builds upon GTOTP [24] and incorporates a Registration Authority (RA) to dynamically manage device nodes, and we combine the overheads of the RA and the server. A series of Pseudo-random Functions are defined in DGTOTP for generating keys, and in the simulation we use SHA-256 instead. RA pre-generates keys and the Merkle tree for the virtual device nodes, and records the mapping relationship between the joined devices and the virtual nodes. The device generates verification credentials (passwords). In the verification stage, the server obtains records from RA, and then verifies the Merkle tree and the OTP.

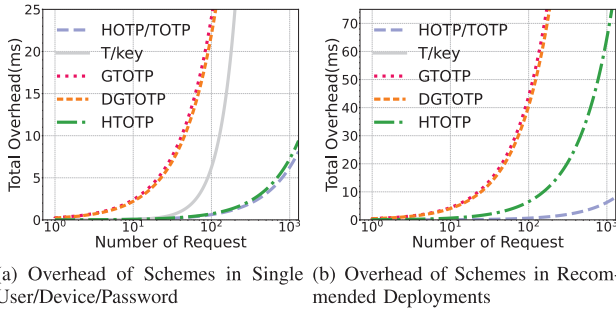


Fig. 6. Comparison of verification overhead between OTP schemes (HOTP [6], TOTP [7], T/Key [8], GTOTP [24], DGTOTP [25], our HTOTP). Figure 6(a) shows the overhead of the solution when handling multiple requests under a single verification condition. Figure 6(b) shows the overhead of the solution when handling multiple requests under their respective recommended deployments. The abscissa represents exponential growth.

In our HTOTP, a variant of ElGamal encryption and decryption is involved during registration, with an overhead of 2 ECC point multiplications. Additionally, in Table IV, we use w to represent the size of honeywords, which generally takes the values of 20 [16] or 40 [3]. From the simulation results, even if the value of w is relatively large, the overheads of registration and verification still remain at a low level. According to the parameter specifications in this section, the overheads are not higher than 0.5 ms. Therefore, the deployment of honeywords can obtain higher security guarantee while not introducing high overhead.

We summarize the overheads of devices and servers during the verification phase. In Figure 6(a), we present the basic overheads. In our HTOTP, the size w of the honeyword is set to 1. In GTOTP [24] and DGTOTP [25], the number d of user devices is set to 1. In T/key [8], $t_1 + t_2$ is set to the number of requests required for each test, and the abscissa grows exponentially. From the overhead curves, it can be seen that the overhead of our scheme for a single operation is comparable to that of the basic HOTP/TOTP [6], [7]. In addition, $t_1 + t_2$ represents the total number of blocks set, which is related to the verification overhead. Therefore, the slope of its overhead curve is higher than that of other curves as the abscissa increases.

Figure 6(b) shows the comparison of the scheme costs under deployments based on recommended settings. In HTOTP, the size w of the honeyword is set to 20. In GTOTP [24] and DGTOTP [25], the number d of user devices is set to 32 (the average value obtained during the evaluation). In T/key [8], $t_1 + t_2$ is set to $k = 2 \times 10^6$. The overhead in the initial verification is approximately 2 seconds, which exceeds the range of the coordinate system. Subsequently, the overhead will decrease as the chain length reduces. The overhead of our solution increases as the honeyword grows, and the overhead curve is in line with Figure 5.

VII. CONCLUSION

In this work, we first revisit academic OTP schemes and industrial solutions, demonstrating that the current schemes and solutions are unsatisfactory due to several vulnerabilities, including security threats arising from loosely coupled OTP architectures, device compromises, and OTP forgery. Subsequently, we construct a tightly coupled password-OTP

two-factor architecture to address device compromises and OTP forgery. Additionally, we incorporate honeywords into this tightly coupled architecture to counter server compromises and online password guessing attacks. For the first time, our scheme extends the security of OTP schemes to provide resistance against server/device compromises and OTP factor forgery. Specifically, we present a comprehensive set of security attributes and application metrics to analyze OTP schemes. We provide proofs and probabilistic analyses of the scheme's security. This work is expected to contribute to the development of more secure OTP authentication in 2FA/MFA.

REFERENCES

- [1] D. Malone and K. Maher, "Investigating the distribution of password choices," in *Proc. 21st Int. Conf. World Wide Web*, Apr. 2012, pp. 301–310.
- [2] D. Wang, H. Cheng, P. Wang, X. Huang, and G. Jian, "Zipf's law in passwords," *IEEE Trans. Inf. Forensics Security*, vol. 12, no. 11, pp. 2776–2791, Nov. 2017.
- [3] D. Wang, Y. Zou, Q. Dong, Y. Song, and X. Huang, "How to attack and generate honeywords," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2022, pp. 966–983.
- [4] L. Lamport, "Password authentication with insecure communication," *Commun. ACM*, vol. 24, no. 11, pp. 770–772, 1981.
- [5] N. M. Haller, *The S/KEY One-Time Password System*, document RFC 1760, 1995. [Online]. Available: <https://www.rfc-editor.org/info/rfc1760>
- [6] D. M'Raihi, D. M'Raihi, F. Hoornaert, D. Naccache, M. Bellare, and O. Ranen, *HOTP: An HMAC-Based One-Time Password Algorithm*, document RFC 4226, 2005. [Online]. Available: <https://www.rfc-editor.org/info/rfc4226>
- [7] D. M'Raihi, J. Rydell, M. Pei, and S. Machani, *TOTP: Time-Based One-Time Password Algorithm*, document RFC 6238, 2011. [Online]. Available: <https://www.rfc-editor.org/info/rfc6238>
- [8] D. Kogan, N. Manohar, and D. Boneh, "T/key: Second-factor authentication from secure hash chains," in *Proc. ACM CCS*, 2017, pp. 983–999.
- [9] Google.(Sep. 2020). *Google Authenticator Opensource*. [Online]. Available: <https://github.com/google/google-authenticator>
- [10] J. Hindy. (Oct. 2024). *Microsoft Authenticator: What it is, How it Works, and How to Use it!* Android Authority. [Online]. Available: <https://www.androidauthority.com/microsoft-authenticator-987754/>
- [11] Google.(Mar. 2024). *Get Verification Codes With Google Authenticator*. [Online]. Available: <https://support.google.com/accounts/answer/1066447?hl=en&co=GENIE.Platform%3DAndroid>
- [12] Z. Lei, Y. Nan, Y. Fratanionio, and A. Bianchi, "On the insecurity of SMS one-time password messages against local attackers in modern mobile devices," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2021, pp. 1–18.
- [13] J. Zhao, B. Ding, Y. Guo, Z. Tan, and S. Lu, "SecureSIM: Rethinking authentication and access control for SIM/eSIM," in *Proc. 27th Annu. Int. Conf. Mobile Comput. Netw.*, Oct. 2021, pp. 451–464.
- [14] S. Khandelwal. (May 2017). *Real-world SS7 Attack—Hackers are Stealing Money From Bank Accounts*. [Online]. Available: <https://thehackernews.com/2017/05/ss7-vulnerability-bank-hacking.html>
- [15] NIST. (Jun. 2017). *NIST Special Publication 800-63B: Digital Identity Guidelines*. [Online]. Available: <https://nvlpubs.nist.gov/nistpubs/specialpublications/nist.sp.800-63b.pdf>
- [16] A. Juels and R. L. Rivest, "Honeywords: Making password-cracking detectable," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Berlin, Germany, 2013, pp. 145–160.
- [17] E. Erdem and M. T. Sandikkaya, "OTPaas—One time password as a service," *IEEE Trans. Inf. Forensics Security*, vol. 14, no. 3, pp. 743–756, Mar. 2019.
- [18] D. H. Krawczyk, M. Bellare, and R. Canetti, *HMAC: Keyed-Hashing for Message Authentication*, document RFC 2104, 1997. [Online]. Available: <https://www.rfc-editor.org/info/rfc2104>
- [19] P. J. N. II, M. Straw, C. Metz, and N. M. Haller, *A One-Time Password System*, document RFC 2289, 1998. [Online]. Available: <https://www.rfc-editor.org/info/rfc2289>
- [20] C. J. Mitchell and L. Chen, "Comments on the S/KEY user authentication scheme," *ACM SIGOPS Operating Syst. Rev.*, vol. 30, no. 4, pp. 12–16, Oct. 1996.

- [21] M. H. Eldefrawy, M. K. Khan, K. Alghathbar, T. Kim, and H. Elkamchouchi, "Mobile one-time passwords: Two-factor authentication using mobile phones," *Secur. Commun. Netw.*, vol. 5, no. 5, pp. 508–516, May 2012.
- [22] L. Gong, J. Pan, B. Liu, and S. Zhao, "A novel one-time password mutual authentication scheme on sharing renewed finite random sub-passwords," *J. Comput. Syst. Sci.*, vol. 79, no. 1, pp. 122–130, Feb. 2013.
- [23] C.-S. Park, "One-time password based on hash chain without shared secret and re-registration," *Comput. Secur.*, vol. 75, pp. 138–146, Jun. 2018.
- [24] Z. Yang, C. Jin, J. Ning, Z. Li, A. Dinh, and J. Zhou, "Group time-based one-time passwords and its application to efficient privacy-preserving proof of location," in *Proc. Annu. Comput. Secur. Appl. Conf.*, Dec. 2021, pp. 497–512.
- [25] X. Cao et al., "Dynamic group time-based one-time passwords," *IEEE Trans. Inf. Forensics Security*, vol. 19, pp. 4897–4913, 2024.
- [26] H. Sun, K. Sun, Y. Wang, and J. Jing, "TrustOTP: Transforming smartphones into secure one-time password tokens," in *Proc. 22nd ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2015, pp. 976–988.
- [27] C. Peeters, C. Patton, I. N. S. Munyaka, D. Olszewski, T. Shrimpton, and P. Traynor, "SMS OTP security (SOS): Hardening SMS-based two factor authentication," in *Proc. ACM Asia Conf. Comput. Commun. Secur.*, May 2022, pp. 2–16.
- [28] Yubico.(Mar. 2024). *Yubico OTP*. [Online]. Available: <https://docs.yubico.com/yesdk/users-manual/application-otp/yubico-otp.html>
- [29] OKTA. (Mar. 2025). *MFA Factor Configuration*. [Online]. Available: <https://help.okta.com/en-us/content/topics/security/mfa/yubikey.htm>
- [30] P. Identity. *Authenticating With Pingid Using a Yubikey*. [Online]. Available: https://docs.pingidentity.com/r/en-us/pingid-user-guide/pid Ug_auth_using_a_yubikey
- [31] Secret Double Octopus. (Mar. 2025). *Empower Your Yubikeys With Secret Double Octopus's Passwordless Enterprise*. [Online]. Available: <https://doubleoctopus.com/technology-partners/yubico/>
- [32] Yubico.(Mar. 2025). *OATH Walk-Through*. [Online]. Available: https://developers.yubico.com/OATH/OATH_Walk-Through.html
- [33] Google.(Sep. 2020). *Google Authenticator OpenSource*. [Online]. Available: <https://github.com/google/google-authenticator>
- [34] Authy. (Mar. 2025). *Authy Introduction*. [Online]. Available: <https://authy.com/features/>
- [35] Lastpass. (Nov. 2024). *Generate One-Time Passwords for Last-Pass*. [Online]. Available: <https://support.lastpass.com/help/how-do-i-generate-temporary-one-time-passwords-for-lastpass>
- [36] 1password. (Mar. 2025). *Use 1Password as an Authenticator for Sites With Two-Factor Authentication*. [Online]. Available: <https://support.1password.com/one-time-passwords/>
- [37] Duo Mobile. (Mar. 2025). *Duo Mobile Application: Secure Access from Your Smartphone*. [Online]. Available: <https://duo.com/product/multi-factor-authentication-mfa/duo-mobile-app>
- [38] N. Chakraborty and S. Mondal, "Few notes towards making honeyword system more secure and usable," in *Proc. 8th Int. Conf. Secur. Inf. Netw.*, Sep. 2015, pp. 237–245.
- [39] I. Erguler, "Achieving flatness: Selecting the honeywords from existing user passwords," *IEEE Trans. Depend. Secure Comput.*, vol. 13, no. 2, pp. 284–295, Mar. 2016.
- [40] D. Wang, H. Cheng, P. Wang, J. Yan, and X. Huang, "A security analysis of honeywords," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2018, pp. 1–15.
- [41] Akshima, D. Chang, A. Goel, S. Mishra, and S. K. Sanadhya, "Generation of secure and reliable honeywords, preventing false detection," *IEEE Trans. Depend. Secure Comput.*, vol. 16, no. 5, pp. 757–769, Sep. 2019.
- [42] W. Li, P. Wang, and K. Liang, "HPAKE: Honey password-authenticated key exchange for fast and safer online authentication," *IEEE Trans. Inf. Forensics Security*, vol. 18, pp. 1596–1609, 2023.
- [43] S. Jarecki, H. Krawczyk, and J. Xu, "OPAQUE: An asymmetric PAKE protocol secure against pre-computation attacks," in *Proc. EUROCRYPT*, 2018, pp. 456–486.
- [44] D. Wang and P. Wang, "Two birds with one stone: Two-factor authentication with security beyond conventional bound," *IEEE Trans. Depend. Secure Comput.*, vol. 15, no. 4, pp. 708–722, Jul./Aug. 2018.
- [45] Dallas, TX, USA. (Jul. 2024). *Zimperium Uncovers Sophisticated SMS Stealer Campaign: Android-targeted Malware Enables Corporate Network and Application Infiltration*. [Online]. Available: <https://www.zimperium.com/resources/zimperium-uncovers-sophisticated-sms-stealer-campaign-android-targeted-malware-enables-corporate-network-and-application-infiltration/>
- [46] D. Dolev and A. C. Yao, "On the security of public key protocols," *IEEE Trans. Inf. Theory*, vol. IT-29, no. 2, pp. 198–208, Mar. 1983.
- [47] S. Rajamanickam, S. Vollala, R. Amin, and N. Ramasubramanian, "Insider attack protection: Lightweight password-based authentication techniques using ECC," *IEEE Syst. J.*, vol. 14, no. 2, pp. 1972–1983, Jun. 2020.
- [48] D. Wang, Q. Gu, X. Huang, and P. Wang, "Understanding human-chosen PINs: Characteristics, distribution and security," in *Proc. ACM Asia Conf. Comput. Commun. Secur. (ASIA CCS)*, Abu Dhabi, United Arab Emirates, 2017, pp. 372–385.



Zixuan Ding is currently pursuing the Ph.D. degree with the College of Cyber Science, Nankai University, Tianjin, China. He has published papers at venues, such as IEEE TRANSACTIONS ON INTELLIGENT TRANSPORTATION SYSTEMS, IEEE INTERNET OF THINGS JOURNAL, IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY, and *Journal of Systems Architecture*. His research interests include applied cryptography, authentication protocol, and password-based authentication.



Ding Wang received the Ph.D. degree in information security from Peking University in 2017. He was supported by the "Boya Postdoctoral Fellowship" with Peking University from 2017 to 2019. Currently, he is a Full Professor with Nankai University. As the first author (or corresponding author), he has published more than 100 papers at venues, such as IEEE S&P, ACM CCS, NDSS, USENIX Security, IEEE TRANSACTIONS ON DEPENDABLE AND SECURE COMPUTING, and IEEE TRANSACTIONS ON INFORMATION FORENSICS AND SECURITY. His main research interests include passwords, authentication, and provable security. His research has been reported by over 200 media, such as The Wall Street Journal, Daily Mail, Forbes, IEEE Spectrum, and Communications of the ACM, appeared in the Elsevier 2017 "Article Selection Celebrating Computer Science Research in China," and resulted in the revision of the authentication guideline NIST SP800-63-2. He has been involved in the community as the PC Chair/TPC Member for over 60 international conferences, such as NDSS 2023–2025, ACM CCS 2022, PETS 2022–2024, ACSAC 2020–2024, RAID 2024/2023, IEEE EuroS&P 2025, FC 2025, ACM AsiaCCS 2025/2022/2021, ICICS 2018–2025, and SPNCE 2020–2022. He received the ACM China Outstanding Doctoral Dissertation Award, the Best Paper Award from INSCRYPT 2018, the First Prize of Cryptologic Innovation Award from China Association for Cryptologic Research, and the First Prize of Natural Science Award from the Ministry of Education.