Strong Authentication without Tamper-Resistant Hardware and Application to Federated Identities

Zhenfeng Zhang\textsuperscript{1,2,3}, Yuchen Wang\textsuperscript{1,2} and Kang Yang\textsuperscript{4}
\textsuperscript{1}TCA Lab of State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences
\textsuperscript{2}University of Chinese Academy of Sciences
\textsuperscript{3}The Joint Academy of Blockchain Innovation
\textsuperscript{4}State Key Laboratory of Cryptology
zhenfeng@iscas.ac.cn, wangyuchen@tca.iscas.ac.cn, yangk@sklc.org

Abstract—Shared credential is currently the most widespread form of end user authentication with its convenience, but it is also criticized for being vulnerable to credential database theft and phishing attacks. While several alternative mechanisms are proposed to offer strong authentication with cryptographic challenge-response protocols, they are cumbersome to use due to the need of tamper-resistant hardware modules at user end.

In this paper, we propose the first strong authentication mechanism without the reliance on tamper-resistant hardware at user end. A user authenticates with a password-based credential via generating designated-verifiable authentication tokens. Our scheme is resistant to offline dictionary attacks in spite that the attacker can steal the password-protected credentials, and thus can be implemented on general-purpose devices.

More specifically, we first introduce and formalize the notion of Password-Based Credential (PBC), which models the resistance of offline attacks and the unforgeability of authentication tokens even if attackers can see authentication tokens and capture password-wrapped credentials of honest users. We then present a highly-efficient construction of PBC using a “randomize-then-prove” approach, and prove its security. The construction does not involve bilinear-pairings, and can be implemented with common cryptographic libraries for many platforms. We also present a technique to transform the PBC scheme to be publicly-verifiable, and present an application of PBC in federated identity systems to provide holder-of-key assertion mechanisms. Compared with current certificate-based approaches, it is more convenient and user-friendly, and can be used with the federation systems that employ privacy-preserving measures (e.g., Sign-in with Apple).

We also implement the PBC scheme and evaluate its performance for different applications over various network environments. When PBC is used as a strong authentication mechanism for end users, it saves 26\%-36\% of time than the approach based on ECDSA with a tamper-resistant hardware module. As for its application in federation, it could even save more time when the user proves its possession of key to a Relying Party.

I. INTRODUCTION

Passwords, or shared credentials, have dominated the realm of authentication for several decades, but they have also been regarded as the weakest points of modern computer systems. Traditional shared credential authentication mechanisms require the credentials of users to be stored in centralized repositories at servers, and to be explicitly transferred, which creates notable targets for attackers. They can either be stolen in batches from breached centralized repositories, or be captured while being transferred (i.e., through phishing attacks).

To eliminate the security risks raised by shared credentials, the techniques of strong authentication have been widely adopted by the industry and standardization communities as alternative approaches. In particular, strong authentication schemes are the cryptographic challenge-response identification protocols that one entity (i.e., the claimant) “proves” its identity to another entity (i.e., the verifier) by demonstrating the knowledge of a secret known to be associated with that entity. During authentication, the secret is neither revealed to the verifier nor transferred over the channel. Such mechanisms can be built with symmetric-key or public-key cryptographic primitives. For mechanisms based on symmetric-key crypto, the two entities share a symmetric-key, and the claimant corroborates its identity by demonstrating the knowledge of the shared key by encrypting a challenge, or by generating a MAC (message authentication code) value for the challenge. For mechanisms based on public-key crypto, a claimant demonstrates knowledge of its private key in a way of digitally signing a challenge, or decrypting a challenge encrypted under its public key. As concrete examples, the FIDO Alliance adopts strong authentication with public-key techniques in the Universal Authentication Framework (UAF)\cite{fido}, and the Web Authentication specification of W3C\cite{w3c} also defines an API enabling the use of public key-based credentials by web applications for the purpose of strong authentication.

However, conventional strong authentication mechanisms involve the problem of secure storage of secrets, i.e., secret-keys in symmetric-key cases and private-keys in public-key cases. A common approach is encrypting the secret with a password. This protection is vulnerable to off-line attacks when the attacker can steal the password-protected credential \(\text{[cre]}_{pw}\) and capture the authentication token \(\sigma\) from honest users (e.g., through phishing). It can guess a password \(pw'\) and determine its correctness with off-line manners:

\begin{itemize}
  \item If \(\sigma\) can be verified with the knowledge of \(\text{cre}\) (in the case of symmetric crypto), off-line attacks cannot be avoided since one can guess a \(pw'\) and decrypt \(\text{[cre]}_{pw}\) to obtain
\end{itemize}
be permanently lost once the module is broken or lost. It requires a user to carry, and the secret keys stored by such a module would become another thing to be remembered by the device it resides in. Becomes another thing to be remembered by the device it resides in. The module (i.e., the SIM card and Trusted Platform Module (TPM)) for user end authentication mechanisms, which is also seen as one of the major drawbacks of currently deployed strong authentication mechanisms [17], [47], [43]. The module (i.e., the device it residents) becomes another thing to be remembered to carry, and the secret keys stored by such a module would be permanently lost once the module is broken or lost.

In practice, these secrets are commonly stored with tamper-resistant hardware modules at user end. Both the FIDO UAF protocol [10], [29], [44] and W3C’s specification [60] highly recommend using tamper-resistant hardware modules (e.g., SIM card and Trusted Platform Module (TPM)) for user end implementation (i.e., FIDO authenticator) to protect the private keys and perform cryptographic operations.

The employment of tamper-resistant hardware module decreases the usability of strong authentication schemes as end user authentication mechanisms, which is also seen as one of the major drawbacks of currently deployed strong authentication mechanisms [17], [47], [43]. The module (i.e., the device it residents) becomes another thing to be remembered to carry, and the secret keys stored by such a module would be permanently lost once the module is broken or lost.

For federated identity systems, assertions convey authentication and attribute information of users from Identity Providers (IdPs) to separately-administered Relying Parties (RPs). They can be presented directly to the RP, or be forwarded through the user. Upon receiving an assertion, the RP uses the information contained to identify the user and to make authorization decisions about its accesses to resources controlled by the RP. A bearer assertion can be presented by any party as proof of the bearer’s identity, and may lead to impersonation attacks if it is captured by attackers.

To enhance the security guarantees provided by assertions, the notion of holder-of-key assertion is introduced for many federation approaches (e.g., SAML2 [39] and OAuth2.0 [35]). It prevents assertions from being abused by malicious RPs, and guarantees that the user cannot repudiate for an assertion sent by him. This technique has been adopted for commercial products such as Microsoft’s XBOX [37] and IBM’s Web-Sphere [6], and also utilized by the digital identity guidelines published by NIST [51] as a requirement for the highest level of federation assurance. Particularly, a holder-of-key assertion contains a reference to the key held by the user (e.g., public key). To verify such an assertion, the RP requires the user to cryptographically prove possession of the key (e.g., private key) that corresponds to the key reference presented in the assertion.

Currently, the holder-of-key assertion mechanisms are mainly implemented via certificates [39], [48], where the user must send its X.509 certificate and prove possession of the key referred in the certificate. However, such mechanisms are inconvenient for end users due to the requirement of certificate, and also require tamper-resistant hardware to protect the private keys. Moreover, for federated identity systems which employ RP-specific pseudonyms to protect the privacy of users (e.g., Sign-in with Apple [8]), RPs can collude and link the same user through the key used in holder-of-key assertions, which breaks the un-linkability brought by pseudonyms. With current technologies, an IdP cannot both preserve the privacy of users and support holder-of-key assertions simultaneously.

A. Our Contributions

The main contribution of this paper is the first strong authentication scheme which does not require tamper-resistant hardware modules to protect secrets at user end. We furthermore present its application with federated identity systems to solve the issues of user’s privacy and reliance on tamper-resistant hardware in the current holder-of-key assertion mechanisms. Our scheme wraps (a.k.a., protects) credentials of users to cryptographically prove possession of the key (e.g., private key). To verify such an assertion, the RP requires the user to cryptographically prove possession of the key (e.g., private key) that corresponds to the key reference presented in the assertion.

For federated identity systems, assertions convey authentication and attribute information of users from Identity Providers (IdPs) to separately-administered Relying Parties (RPs). They can be presented directly to the RP, or be forwarded through the user. Upon receiving an assertion, the RP uses the information contained to identify the user and to make authorization decisions about its accesses to resources controlled by the RP. A bearer assertion can be presented by any party as proof of the bearer’s identity, and may lead to impersonation attacks if it is captured by attackers.

To enhance the security guarantees provided by assertions, the notion of holder-of-key assertion is introduced for many federation approaches (e.g., SAML2 [39] and OAuth2.0 [35]). It prevents assertions from being abused by malicious RPs, and guarantees that the user cannot repudiate for an assertion sent by him. This technique has been adopted for commercial products such as Microsoft’s XBOX [37] and IBM’s Web-Sphere [6], and also utilized by the digital identity guidelines published by NIST [51] as a requirement for the highest level of federation assurance. Particularly, a holder-of-key assertion contains a reference to the key held by the user (e.g., public key). To verify such an assertion, the RP requires the user to cryptographically prove possession of the key (e.g., private key) that corresponds to the key reference presented in the assertion.

Currently, the holder-of-key assertion mechanisms are mainly implemented via certificates [39], [48], where the user must send its X.509 certificate and prove possession of the key referred in the certificate. However, such mechanisms are inconvenient for end users due to the requirement of certificate, and also require tamper-resistant hardware to protect the private keys. Moreover, for federated identity systems which employ RP-specific pseudonyms to protect the privacy of users (e.g., Sign-in with Apple [8]), RPs can collude and link the same user through the key used in holder-of-key assertions, which breaks the un-linkability brought by pseudonyms. With current technologies, an IdP cannot both preserve the privacy of users and support holder-of-key assertions simultaneously.

A. Our Contributions

The main contribution of this paper is the first strong authentication scheme which does not require tamper-resistant hardware modules to protect secrets at user end. We furthermore present its application with federated identity systems to solve the issues of user’s privacy and reliance on tamper-resistant hardware in the current holder-of-key assertion mechanisms. Our scheme wraps (a.k.a., protects) credentials of users by passwords, and can be implemented on different devices (e.g., mobile phones and desktop computers) to support cross-terminal authentication, but is resistant to phishing attacks and offline dictionary attacks even in the case that the attacker breaks the un-linkability brought by pseudonyms. With current technologies, an IdP cannot both preserve the privacy of users and support holder-of-key assertions simultaneously.

A. Our Contributions

The main contribution of this paper is the first strong authentication scheme which does not require tamper-resistant hardware modules to protect secrets at user end. We furthermore present its application with federated identity systems to solve the issues of user’s privacy and reliance on tamper-resistant hardware in the current holder-of-key assertion mechanisms. Our scheme wraps (a.k.a., protects) credentials of users by passwords, and can be implemented on different devices (e.g., mobile phones and desktop computers) to support cross-terminal authentication, but is resistant to phishing attacks and offline dictionary attacks even in the case that the attacker breaks the un-linkability brought by pseudonyms. With current technologies, an IdP cannot both preserve the privacy of users and support holder-of-key assertions simultaneously.
cloud server. For authentication, the user uses its password \( pw \) and the password-wrapped credential \([cre]_{pw}\) to create an authentication token \( \sigma \) on a challenge message \( m \). The server authenticates the user with its secret key by checking whether \( \sigma \) is valid on \( m \) w.r.t. \( wid \).

We establish the necessary security requirement of Existential UnForgeability under Chosen Message and chosen Verification queries Attacks (EUF-CMVA) for PBC. Our security model covers a wide range of practical threats, where the attacker can corrupt users adaptively, obtain the password-wrapped credentials of honest users, and see authentication tokens generated by honest users.

- We propose a highly efficient construction of PBC, denoted by \( \Pi_{PBC} \), and prove its security under the cryptographic \( q\)-SDH and \( q\)-DDH assumptions in the random oracle model. \( \Pi_{PBC} \) is resistant to off-line attacks in the aforementioned security model since authentication tokens generated by honest users can only be verified by a designated server. This construction does not rely on bilinear-pairings, and thus can be implemented easily by common cryptographic libraries on many platforms.

- We propose a technique to transform the designated-verifiability of \( \Pi_{PBC} \) to public-verifiability. With this technique, we present an application of PBC in federated identity systems to implement the holder-of-key assertion mechanism, which can be integrated into standardized federation protocols such as SAML2 and OpenID Connect. It enables a user to provide a key reference to the IdP, and prove possession of the corresponding key to the RP using a PBC authenticator. Our approach furthermore provides an option that enables the holder-of-key assertions to be verified while hiding the user’s identity, which can be used in federation systems that have applied privacy-preserving measures such as Sign-in with Apple.

- We evaluate the performances of our schemes over Local Area Network (LAN) and Wide Area Network (WAN) under different latencies, and compare them with the approaches using ECDSA with tamper-resistant hardware modules. For strong authentication, PBC (denoted by AUTH-PBC) saves 26%-36% time than the ECDSA-based strong authentication. For its application as holder-of-key assertion mechanism, PBC speeds the process by 41%-55% compared with the approaches based on ECDSA.

B. Technical Overview

Here, we present the challenges we met and the techniques leveraged to tackle them.

Avoid offline attacks. The major challenge of using passwords to protect credentials is avoiding offline dictionary attacks. The credential should not have any structure character, otherwise off-line attacks are feasible as one can guess a \( pw' \), decrypt \([cre]_{pw'} \) with \( pw' \), and check the character structure of \( cre \). The authentication token \( \sigma \) should not be verified with \( cre \) or publicly-verifiable. Our solution tackle this challenge via two techniques:

- The credentials of our schemes are indistinguishable from random group elements, and thus do not have any structure character for off-line attacks. Furthermore, we apply the password-based encryption proposed by [14] to wrap the credentials with passwords, where the decryptions from any passwords obtain group elements with the same structure characters. With this approach, the attacker can only obtain group elements which are indistinguishable from the real credential when guessing the password.

- The authentication tokens are generated with a “randomize-then-prove” paradigm, and can only be verified by a designated verifier who has the corresponding secret key. The authentication token can neither be verified with \( cre \) nor by the attacker who does not know the secret key. In particular, the “randomize-then-prove” paradigm firstly randomizes a credential and then presents it with zero-knowledge proofs, and has been used for anonymous credential protocols [24], [11], [19]. It does not reveal the knowledge of credential and thus resists offline attacks.

Transform PBC to publicly-verifiable. Holder-of-key assertion mechanisms require the user to prove its possession of a secret to any RP, which is not a problem for asymmetric primitives such as digital signature schemes since they are publicly-verifiable. However, this seems to contradict the proposed scheme, which is designated-verifiable. We solve this dilemma as follows:

- We use the IdP (i.e., designated verifier) as a “converter” that transforms our scheme to be publicly-verifiable. It provides RPs with information that is necessary for verifying the authentication tokens generated by users. To prevent this technique from being abused to perform off-line attacks, we require that the IdP must authenticate the user first, and limit the effect of a transformation within one session.

We also consider a scenario that the user authenticates to RPs using IdP-managed unrelated pseudonyms, for preventing the user’s activities from tracking or profiling when these RPs collude. However, the privacy goal is hard to be achieved when holder-of-key assertions are adopted, since RPs can associate the same user with the references of keys.

- We provide an option to convert the scheme to be public-verifiable while preserving the privacy of users, which enables the user to prove the possession of the credential to the RP without revealing its identity. Any RP can verify the holder-of-key assertions without knowing the references to original credentials (i.e., user identifiers), which avoids the user from being linked across multiple RPs through holder-of-key assertions.

C. Related Work

A series of schemes have been proposed to address the problem of large-scale credential leakage in centralized storage of shared credentials. Several password hardening schemes [27], [52], [42] have been proposed to strengthen the username-password authentication for web service providers, where an external crypto server is adopted to carry out certain cryptographic operations. Furthermore, hardware security modules (e.g., Intel SGX) have also been utilized at server side to harden the storage of credentials by protecting the salt values or secret keys [41], [18].

To prevent phishing attacks, many Multi-Factor Authentication (MFA) mechanisms have been designed [59], [40], [55],
and deployed in practice (e.g., The FIDO Universal 2nd Factor (U2F) Protocol [57]). These schemes use dedicated devices such as mobile phones and FIDO U2F keys to perform the 2nd factor authentication, and their security and usability thus rely on these devices. The 2nd authentication factor will lose once the device is broken or lost, and may also be occupied by the attacker once the device is stolen or breached.

Password hardening and MFA techniques improve the username-password authentication, but do not change the authentication mechanism itself (e.g., the credentials are still transferred explicitly from claimer to verifier) which makes them different from the approaches for strong authentication.

Moreover, Pass2Sign [20] enables the user to sign messages with the help of a server using its password. A Pass2Sign server stores salted hashed values of users’ passwords, and the corresponding salt values are stored at user end devices. For authentication, the user calculates the hash value of its password, the salt and a global query identifier, encrypts the hashed value with the server’s public key and sends it to the server. However, Pass2Sign does not consider the threat scenario that attackers could get authentication tokens (e.g., through phishing attacks). That is, an attacker could first capture the hashed values calculated by the users by disguising as the server via providing the user with the public key of its own, and then steal the salt value stored at user end device. After that, it could obtain the password via offline attacks.

II. Preliminaries

Notation. Throughout this paper, we use λ to denote the security parameter, and |n| to denote the set \{1, ..., n\}. The notation \( x \overset{\$}{\leftarrow} S \) denotes that \( x \) is sampled uniformly at random from a set \( S \). For an algorithm \( A \), \( y \leftarrow A(x) \) denotes the process that runs \( A \) on input \( x \) and outputs \( y \) as output. We say that a function \( f : \mathbb{N} \rightarrow [0, 1] \) is negligible if for any positive \( c \), we have \( f(\lambda) < 1/\lambda^c \) for sufficient large \( \lambda \). For simplicity, we use negl to denote an unspecified negligible function. Let \( G \) be a group of prime order \( p \) generated by \( g \), and \( G^* \) denote \( G \setminus \{1\} \) where 1 is the identity element of \( G \). Finally, \([M]_{pw}\) denotes the ciphertext on a message \( M \) encrypted with a password \( pw \).

A. Cryptographic Assumptions

We recall two intractability assumptions in \( G \).

Definition 1: (q-SDH) [15] We say that the q-Strong Diffie-Hellman (q-SDH) assumption holds in group \( G \), if for all Probabilistic Polynomial Time (PPT) adversaries \( A \) such that

\[
\text{Adv}^{\text{SDH}}_G \overset{\text{def}}{=} \Pr[x \overset{\$}{\leftarrow} Z_p^* : A(g, g^x, ..., g^{x^q}) = (c, g^{1/(x+c)})] < \text{negl}(\lambda), \quad \text{where } c \in Z_p \setminus \{-x\}.
\]

We write \( \text{Adv}^{\text{SDH}}_G(t, q) = \max_A \{\text{Adv}^{\text{SDH}}_G(A)\} \), where the maximum is taken over all adversaries of time complexity at most \( t \) and obtaining \( q + 1 \) group elements.

Definition 2: (q-DDHI) [16] We say that the q-Decisional Diffie-Hellman Inversion (q-DDHI) assumption holds in group \( G \), if for all PPT adversaries such that

\[
\text{Adv}^{\text{DDHI}}_G \overset{\text{def}}{=} \left| \Pr[x \overset{\$}{\leftarrow} Z_p^*: A(g, g^x, ..., g^{x^q}, g^{1/x}) = 1] - \Pr[x \overset{\$}{\leftarrow} Z_p^*, U \overset{\$}{\leftarrow} G^*: A(g, g^x, ..., g^{x^q}, U) = 1] \right| < \text{negl}(\lambda)
\]

We write \( \text{Adv}^{\text{DDHI}}_G(t, q) = \max_A \{\text{Adv}^{\text{DDHI}}_G(A)\} \) where the maximum is taken over all adversaries of time complexity at most \( t \) and obtaining \( q + 2 \) group elements.

B. Non-Interactive Zero-Knowledge Proofs

Non-Interactive Zero-Knowledge Proofs (NIZKs) enable a prover to prove that a statement \( x \) is in a given NP language \( L \) in a zero-knowledge way, where \( L \) is defined by an NP relation \( R \), i.e., \( L = \{x \mid \exists w \text{ s.t. } R(x, w) = 1\} \) and \( w \) is called a witness for \( x \).

Signature proofs of knowledge (SPKs) [23] are NIZK proofs constructed by using the Fiat-Shamir heuristic [28] to transform Sigma protocols in the random oracle model [13]. When referring to the SPKs on proving knowledge of discrete logarithms and statements about them, we will follow the notation introduced by Camenisch et al. [21] to abstract the SPKs. For example, \( \pi \leftarrow \text{SPK}(\{a: g^a = w\})(m) \) denotes a signature proof of knowledge on proving knowledge of a witness \( a \) such that \( u = g^a \), which signs a message \( m \) and outputs a proof \( \pi \) on statement \( x = (g, u) \). Let \( \text{Verify}_{\text{SPK}}((g, u), m, \pi) \) denote the verification algorithm of SPK, which outputs 1 if \( \pi \) is valid on \( m \) w.r.t. a statement \( (g, u) \) and 0 otherwise. The signature proofs of knowledge are zero-knowledge, sound and knowledge extractable in the random oracle model [49].

Security Definitions for NIZK. Informally, an NIZK system \( \Pi \) is said to be unbounded zero-knowledge if there exists a simulator \( \Sigma \) without knowing any witness can simulate the proofs on unbounded number of statements such that no adversary can distinguish the simulated proofs from the real ones created with witnesses. In particular, the adversary has access to an oracle \( O_{\Sigma} \) which returns either the real proofs produced by the prover having witnesses or the simulated proofs generated by \( \Sigma \). We use \( \text{Adv}^{\text{unb}}_{\Pi}(t, q_{sk}) \) to denote the maximum advantage of all adversaries against the unbounded zero-knowledge of \( \Pi \) who run in time \( t \) and make at most \( q_{sk} \) queries to \( O_{\Sigma} \). The soundness of an NIZK system \( \Pi \) assures that no adversary can prove a false statement (i.e., outside \( L \)), where \( \text{Adv}^{\text{sound}}_{\Pi}(t) \) denotes the maximum advantage of all adversaries who run in time \( t \). We say that an NIZK system \( \Pi \) satisfies the knowledge extractability if there exists an extractor \( \text{Ext} \) which can extract a witness from the proof created by the adversary, where \( \text{Adv}^{\text{ext}}_{\Pi}(t) \) denotes the maximum advantage of all adversaries who run in time \( t \). Simulation-sound extractability of an NIZK system \( \Pi \) guarantees that the extractor \( \text{Ext} \) still works for the proof produced by the adversary even if the adversary sees many simulated proofs from a simulator oracle \( O_{\text{sim}} \) and the extracted results on the proofs created by itself from an extractor oracle \( O_{\text{ext}} \). We denote by \( \text{Adv}^{\text{ext}}_{\Pi}(t, q_{sk}, q_{ext}) \) the maximum advantage of all adversaries against the simulation-sound extractability of \( \Pi \) who run in time \( t \) and make at most \( q_{sk} \) and \( q_{ext} \) queries to \( O_{\text{sim}} \) and \( O_{\text{ext}} \) respectively. We refer the reader to [32, 9] for the formal definitions of the above security properties.
### III. Password-Based Credential

In this section, we propose a new cryptographic primitive called Password-Based Credential (PBC), which protects a user’s credential issued by a server with a password and uses both the password-protected credential and password to generate authentication tokens to authenticate to the server with a challenge-response procedure, and formalize its security considering practical threats. Our model is game-based, and inspired by the security model for PAKE protocols [12].

We define a security notion of Existential Unforgeability under Chosen Message and chosen Verification queries Attack (EUF-CMVA) for PBC schemes. Informally, EUF-CMVA guarantees that no adversary can forge valid authentication token on a fresh message for an honest user. A PBC scheme is EUF-CMVA secure means that this scheme is secure as long as the number of users, the oracles are defined as follows:

- **REVEALCRED(i)**: If $i \in [n]$, this oracle outputs the password-wrapped credential $[\text{cre}_i]_{pw_i}$ of user $i$ and adds $i$ to RUcred.
- **REVEALPW(i)**: If $i \in [n]$, this oracle outputs the password $pw_i$ of user $i$ and adds $i$ to RUpw.
- **SIGN(i, m)**: If $i \in [n]$ and $i \notin RUpw \cap RUcred$, this oracle returns $\sigma \leftarrow \text{Sign}(\text{uid}_i, \text{pw}_i, [\text{cre}_i]_{pw_i}, m)$ and adds $(i, m)$ to the set $Q$.
- **VERIFY(uid, m, s)**: this oracle returns the verification result on an authentication token $\sigma$ and challenge message $m$ by running Verify(sk, uid, m, s).

Each oracle represents a threat scenario that may occur in practice. In the following, we describe the motivations behind oracles, which also include the abilities and assumptions we adopted for real-life attackers.

- **REVEALCRED**: We allow the adversary to obtain the password-wrapped credentials of its chosen users via making queries to the REVEALCRED oracle. In our oracle, we model the case that the password-wrapped credentials are stored as a file on a general purpose device, and may be stolen by malwares [3], through the lost or stolen of the device [4], or even via the automatical backup service performed by a breached cloud server [2].
- **REVEALPW**: The adversary could also reveal the passwords of users on its choices by having access to the REVEALPW oracle. This corresponds to the case that the user encrypts its credential with a password that has already leaked.
- **SIGN**: When a PBC scheme is adopted for end user authentication, the adversary may see many authentication tokens. This ability is modeled by the SIGN oracle, which allows the adversary to obtain authentication tokens of its chosen users via making queries to the SIGN oracle. We model this oracle since attackers in practice can capture the authentication tokens when they are transferred from the end users to servers (e.g., via phishing attacks).
- **VERIFY**: Furthermore, the adversary can get the verification result of an authentication token $\sigma$ and a message $m$ via impersonating a user to interact with the server. We model this attack ability with the VERIFY oracle, which allows the adversary to obtain the decision result on any query (username, message, token) made by it. In this oracle, we model real-world attacker’s ability of communicating with the server. We do not assume that the attacker can break
The Goal of Adversary. The goal of adversary is to forge a valid authentication token on a fresh message that has never been queried to the SIGN oracle for an honest user, which means that it can pretend as the honest user by authenticating to the server. To define the adversary’s goal formally, we need to eliminate some trivial cases and to bound the advantage of the adversary as follows:

In the game of EUF-CMVA security, the adversary attempts to generate an authentication token on an username $uid^*$ (corresponding to an unregistered user or honest user $i^*$) and a fresh message $m^*$. The adversary is not allowed to trivially reveal both the password-wrapped credential and password of user $i^*$, or answers with a $(i^*, m^*, σ^*)$ where $σ^*$ is obtained by querying SIGN$(i^*, m^*)$. We consider two cases when bounding the advantage of the adversary.

1) In the first case that either $uid^*$ has not been registered or the password-wrapped credentials of user $i^*$ is not revealed, the adversary’s advantage should be negligible. That is to say, it is impossible for an attacker to impersonate as an honest user and thus is impossible to access to the password-wrapped credential.

2) In the second case that the password-wrapped credential $\{cre_{i^*}\}_{pw^*}$ of user $i^*$ has been revealed, the adversary can mount online dictionary attacks with the following steps:
   a) Guess a password $pw'$ and use $(uid^*, pw', [cre_{i^*}]_{pw^*})$ to generate a token $σ'$ on an arbitrary message $m'$
   b) Make a query with $(uid^*, m', σ')$ as the input to the VERIFY oracle.

In this case, the adversary’s advantage is bounded by the standard advantage of password-based authentication protocols, which means that the attacker cannot do any better than guessing the password online.

**Definition 4 (EUF-CMVA):** We say that a PBC scheme PBC is EUF-CMVA secure, if for all PPT adversaries $\mathcal{A}$, any polynomial-size integer $n$, any $n$ different usernames $\{uid_i\}_{i=1}^n \in \mathcal{U}$ such that the following holds:

\[
\begin{align*}
\text{Adv}^\text{PBC-case-1}_{\text{EUF-CMVA}}(\mathcal{A}) &\triangleq \Pr[\text{Exp}_{\text{PBC}}^\text{PBC-case-1}(\mathcal{A}) = 1] \leq \text{negl}(\lambda), \\
\text{Adv}^\text{PBC-case-2}_{\text{EUF-CMVA}}(\mathcal{A}) &\triangleq \Pr[\text{Exp}_{\text{PBC}}^\text{PBC-case-2}(\mathcal{A}) = 2] \leq \frac{q_{v}}{|\mathcal{D}|} + \text{negl}(\lambda),
\end{align*}
\]

where $\text{Exp}_{\text{PBC}}^\text{PBC}(\mathcal{A})$ is defined in Fig. 3 and $q_v$ is the number of queries made by $\mathcal{A}$ to the VERIFY oracle.

Similar to the security notion of digital signatures [30], a stronger variant of the EUF-CMVA security is strongly Existential UnForgeability under Chosen Message and chosen Verification queries Attack (sEUF-CMVA), meaning that a forgery $(m^*, σ^*)$ is considered as valid if $(m^*, σ^*)$ is not from the SIGN oracle. The security definition of sEUF-CMVA is easy to be obtained by slightly modifying the definition of EUF-CMVA, i.e., $Q$ now additionally includes the authentication tokens created by the SIGN oracle and $\text{Exp}_{\text{PBC}}^\text{EUF-CMVA}(\mathcal{A})$ returns 0 if $(i^*, m^*, σ^*) \in Q$. The adversary’s advantage $\text{Adv}^\text{PBC-case-i}_{\text{EUF-CMVA}}(\mathcal{A})$ for $\forall i \in \{1, 2\}$ is defined in the same way as in Definition 4.

IV. A PRACTICAL CONSTRUCTION OF PBC

We propose a practical PBC scheme denoted by $\Pi_{\text{PBC}}$, which satisfies the security definition of sEUF-CMVA under the $q$-SDH and $q$-DDH assumptions. It is efficient enough to be deployed in practice, and can be implemented by common cryptographic libraries in many programming languages (e.g., OpenSSL in C/C++ and Bouncy Castle in Java) with standardized elliptic curves. In this section, we will first give the basic ideas underlying the construction of $\Pi_{\text{PBC}}$, then present the detailed protocol as well as prove its security under the security model described in Section III and finally present the application of PBC as a strong authentication mechanism for end user authentication.

A. High Level Description

In our concrete construction, the server creates a tag on $uid$ with its secret key as the user’s credential $cre$, which is then encrypted by the user with its password $pw$. In the process of authentication, the user proves its possession of $cre$ with respect to $uid$, and sends $σ$ and $uid$ to the server. To avoid offline attacks when the attacker could see many authentication tokens and password-wrapped credentials, the techniques that we adopted are explained as follows:

- We use $g^{1/(γ+uid)}$ as the credential of user, where $γ$ is the secret key held by server. This is inspired by [16]. The credential is indistinguishable from random group elements without the knowledge of $γ$. We encrypt the credential $cre$ with $pw$ by $[cre]_{pw} \leftarrow cre \cdot H_2(pw)$, where the corresponding decryption algorithm is $cre \leftarrow [cre]_{pw} \cdot H^{-1}_2(pw)$ and $H_2 : \mathcal{D} \rightarrow \mathcal{G}$ is a cryptographic hash function. An attacker who captures $[cre]_{pw}$ can guess the password with $pw'$ and only obtain $[cre]_{pw'} \cdot H_2(pw')$. However, it is an group element that is indistinguishable from the real credential $cre$, which can not be used by the attacker to decide whether $pw'$ is the correct password.

- We leverage the “randomize-then-prove” paradigm when the user proves its possession of a credential w.r.t. $uid$. The user
randomizes its credential $cre$ as $T$ with a randomness $a$ (i.e., $T = cre^a$). It then proves the validity of the randomized credential $T$ using a signature proof of knowledge $\pi_T = SPK\{(a) : g^a = PK\}(m)$ for an implicit public-key $PK$.

- To verify an authentication token $\sigma = (T, \pi_T)$, the server first computes $PK = T^{\gamma + wid}$, and then checks the validity of $\pi_T$. If valid, the server is assured that the claimer holds the secret $a$ such that $g^a = T^{\gamma + wid}$, and then believes that it is the legitimate user who has been issued the credential w.r.t. $uid$, since $T^{-a}$ has exactly the form $g^{1/(\gamma + wid)}$.

- The point here is that $\sigma$ only could be verified by the server who issued the credential (i.e., holding $\gamma$), since only the server can obtain the complete statement $(T, PK)$ to be proved in the signature proof of knowledge. Furthermore, it is also infeasible for an attacker to check the validity of $\pi_T$ with $cre$, since $PK$ can not be derived from $T$ and $cre$.

With these approaches, we eliminate the possibility of offline attacks even if an attacker has the ability to steal the password-wrap credentials. The attacker can neither verify the correctness of its guesses on the password through the decryption results directly, nor by checking the correctness of the authentication token generated by it as in the conventional strong authentication mechanisms.

We note that a PBC scheme can not be constructed with undeniable signatures [25] or designated verifier signatures [34, 5].

**B. The Detailed Construction**

Let $H_G : \mathcal{D} \rightarrow \mathbb{G}$ be a random oracle. Our scheme $\Pi_{SPK}$ is constructed as follows:

- **Setup($1^\lambda$):** Given a security parameter $\lambda$, the setup algorithm chooses a set of group parameters $(\mathbb{G}, p, g)$ such that $p$ is an at least $2\lambda$-bit prime, and then outputs $pp = (\mathbb{G}, p, g)$.

- **KeyGen$(pp)$:** Given the public parameters $pp$, a server runs the key generation algorithm which picks $\gamma \xleftarrow{\$} \mathbb{Z}_p^*$ and computes $w \leftarrow g^\gamma$. The server sets $sk \leftarrow \gamma$ and publishes $isp \leftarrow w$, and then initializes $Reg$ as empty.

- **Issue$($uid, pw$)$:** The execution of the secure channel the channel could be established following standard approaches, such as the TLS protocol.

1. A user sends its username $uid$ to the server.
2. The server aborts if $uid \notin Reg$. Otherwise it computes $A \leftarrow g^{1/(\gamma + wid)}$ and adds $uid$ to $Reg$. Then it sends $cre = A$ to the user.
3. The user encrypts its credential $cre = A$ by computing $[A]_{pw} = A \cdot H_G(pw)$, and then stores $[A]_{pw}$.

- **Sign$(uid, pw, [A]_{pw}, m)$:** This algorithm decrypts $[A]_{pw}$ by computing $A \leftarrow [A]_{pw} / H_G(pw)$. Then, it chooses $a \xleftarrow{\$} \mathbb{Z}_p^*$ and randomizes $A$ as $T \leftarrow A^a$ (i.e., $T = g^{a/(\gamma + wid)}$), and generates a signature proof of knowledge w.r.t. $T$ as $\pi_T \leftarrow SPK\{(a) : g^a = T^{\gamma + wid}\}(m)$.

Finally, it outputs an authentication token $\sigma = (T, \pi_T)$.

- **Verify$($uid, $w$, $m$, $\sigma$):** The verification algorithm parses $\sigma$ as $(T, \pi_T)$. If $T = 1$, it outputs 0. Otherwise, it outputs 1 if $\text{Verify}_{SPK}\{(g, T, uid, \gamma), m, \pi_T\} = 1$ and 0 otherwise.

For practical usage where the user may want to register and authenticate with the username on its own choice, $uid$ could be generated via a cryptographic hash function $H_p : \{0, 1\}^* \rightarrow \mathbb{Z}_p$, with the chosen username as input.

**Instantiation of SPK.** Below, we give efficient instantiation for the signature proofs of knowledge SPK. Let $H : \{0, 1\}^* \rightarrow \mathbb{Z}_p$ be a random oracle, SPK could be efficiently instantiated as follows:

**Prove:** Pick $r \xleftarrow{\$} \mathbb{Z}_p$ and compute $R \leftarrow g^r$. Then, compute $c \leftarrow H(g, T, uid, R, m)$. Next, compute $s \leftarrow r + c \cdot a \mod p$. Finally, output a proof $\pi = (c, s)$.

**Verify:** Given a tuple $(g, T, uid, \gamma)$, a message $m$ and a proof $\pi = (c, s)$, compute $R' \leftarrow g^c \cdot T^{-(\gamma + wid) \cdot c \mod p}$, and then calculate $\delta' \leftarrow H(g, T, uid, R', m)$. Output 1 if $\delta' = c$ and 0 otherwise.

One could easily observe that the instantiation of SPK is an application of the Schnorr signature scheme [54], where the public key $T^{uid+\gamma}$ could only be computed by the server who has $\gamma$. Furthermore, the Sign and Verify algorithms could also be constructed by other standardized signature algorithms (e.g., ISO/IEC 14888-3 [5]) in the same way. For the sake of simplicity, we only present the approach based on Schnorr proof, which obtains optimized efficiency and could be implemented by commonly adopted cryptographic libraries with standard curves.

**C. Security Proof**

In this section, we prove that our scheme $\Pi_{SPK}$ is sEUF-CMVA secure, provided that $q$-SDH and $q$-DDHI assumptions hold in $\mathbb{G}$, $H_G$ is a random oracle, and SPK is unbounded zero-knowledge and simulation-sound extractable. Furthermore, SPK is zero-knowledge by programming the random oracle [49], and is simulation-sound extractable in the random oracle model [13] and generic group model [55] following along the lines of [53, 56, 24, 61].

**Theorem 1:** Let $A$ be an adversary against the sEUF-CMVA security of PBC scheme $\Pi_{SPK}$ who runs in time $t$, and makes $q_s$ queries to the $\text{SIGN}$ oracle and $q_v$ queries to the $\text{VERIFY}$ oracle. Then, we have:

$$\text{Adv_{\Pi_{SPK}}^{sEUF-CMVA}}(A) \leq \text{Adv_{SPK}}^{\Delta EKF}(q_s, q_v) + (q_v + 1) \text{Adv_{DDHI}}^{\Delta EKF}(t', n + 1) + n \text{Adv_{SDH}}^{\Delta EKF}(t', n),$$

$$\text{Adv_{\Pi_{SPK}}^{sEUF-CMVA}}(A) \leq \frac{q_v}{|D|} \text{Adv_{SPK}}^{\Delta EKF}(q_s, q_v) + (q_v + 1) \text{Adv_{DDHI}}^{\Delta EKF}(t', n + 1) + n \text{Adv_{DDHI}}^{\Delta EKF}(t', n),$$

where $\Delta EKF(t', q_s, q_v) = O(\Delta_{\text{SPK}}^{\Delta EKF}(t', q_s) + \Delta_{\text{SPK}}^{\Delta EKF}(t', q_s, q_v))$, $t' = t + O((q_s + q_v) t_{\exp}), t'' = O(t + n^2 t_{\exp})$, and $t_{\exp}$ denotes the time for one exponentiation.
Proof: This proof will proceed via a sequence of games. We will bound the decrease of $A$’s advantages between two successive games, and denote $A$’s advantage in $G_i$ by $\text{Adv}_i(A)$.

**Game 0.** This is the real game. The adversary $A$ is given pp,isp and a set of usernames $\{uid_i\}_{i=1}^n$, and has access to four oracles $\text{SIGN}, \text{VERIFY}, \text{REVEALPW}, \text{REVEALCRED}$. Finally, $A$ outputs a valid forgery $(\text{uid}^*, m^*, \sigma^*)$. Let $\{(\text{uid}_i, A_i)\}_{i=1}^n$ be a set including the username and credential pairs of $n$ users created in this game. Without loss of generality, we assume that the output of the adversary $(\text{uid}^*, m^*, \sigma^*)$ has been queried to the $\text{VERIFY}$ oracle. We have

$$\text{Adv}^{\text{SEUF-CMVA}}_{\text{inc,case}\cdot i}(A) = \text{Adv}_0(A)$$
for all $i \in \{1, 2\}$.

**Game 1.** This game is the same as Game 0, except the following differences:

- Use a zero-knowledge simulator $\text{Sim}$ to generate the proofs of SPK for the answers of the $\text{SIGN}$ oracle.
- In every $\text{VERIFY}(\text{uid}, m, \sigma)$ query such that $\sigma = (T, \pi_T)$, execute as follows:
  - If $(m, \sigma)$ is from the $\text{SIGN}$ oracle, return 1.
  - If $T = 1$ or $\pi_T$ is not the correct form, return 0.
  - Otherwise, use a knowledge extractor $\text{Ext}$ for SPK to extract a witness $a$ from proof $\pi_T$. If Ext fails, return 0.
- Otherwise, compute $A \leftarrow T^{1/a}$ and return 1 if and only if $A^{\pi_{\text{uid}} + a} = g$. Note that $a \neq 0$ unless $\text{uid} + \gamma \equiv 0 \mod p$ which clearly breaks the discrete-logarithm assumption implied by the assumptions in Theorem 1.

Game 1 behaves exactly like Game 0, except for the simulation of SPK and failing for the extraction of SPK. From the unbounded knowledge and simulation-sound extractability of SPK, we have $\text{Adv}_0(A) \leq \text{Adv}_1(A) + O(\text{Adv}_{\text{SPK}}(T, q_3)) + \text{Adv}_{\text{SPK}}(T, q_3, q_0))$.

**Game 2.** This game is the same as Game 1, except that for every $\text{SIGN}(\ast, \ast)$ query, pick $u \leftarrow Z_p^*$, compute $T \leftarrow g^u$. Then, compute $V \leftarrow w^u$. $T^{\ast}$, use $\text{Sim}$ to generate a proof $\pi_T$ on a statement $(g, T, V)$, and respond with $\sigma \leftarrow (T, \pi_T)$.

The element $T$ in Game 2 has the same distribution as the one in Game 1. So, we have $\text{Adv}_1(A) = \text{Adv}_2(A)$.

**Game 3.** This game is the same as Game 2, except that for each $\text{VERIFY}(\text{uid}', m, \sigma)$ query with $\text{uid}' \in U$, when an element $A$ is computed with either a witness $a$ extracted by Ext for $\text{uid}' = \text{uid}$, changing the verification manner of $\sigma$ as: Return 0 if $(\text{uid}, A) \notin \{(\text{uid}_i, A_i)\}_{i=1}^n$ and 1 otherwise. We can bound the difference between Game 3 and Game 2 using a reduction $B$ from the $g$-SDH assumption. For the reduction, we use the following lemma.

**Lemma 1:** For all adversaries $B$ running in time $t'$, $B$ aims to win in the experiment as described in Fig. 3 (i.e., $\text{Exp}_2(B)$ outputs 1). Then, $B$’s advantage $\text{Adv}^{\ast\cdot}(t', n)$ is bounded by $(q_v + 1) \text{Adv}^{\ast\cdot}_{\text{SDH}}(O(t' + n^2 \text{exp}), n)$, where $q_v$ is the number of queries to the Check($\gamma$, ·) oracle.

**Proof for Lemma 1:** If there exists an adversary $B$ that makes $q_v$ queries to Check and makes $\text{Exp}_2$ output 1 with probability $\epsilon$, then we can construct an algorithm $B^{\ast\cdot}$ that breaks the $g$-SDH assumption with probability $\epsilon/(q_v + 1)$ by interacting with $B$ as follows:

Given a $g$-SDH instance $(g, g^\gamma, \ldots, g^{\gamma^n}) \in (G^*)^{n+1}$ for some unknown $\gamma \in Z_p^*$, $B^{\ast\cdot}$ aims to output a $(m^*, g^{1/(\gamma + m^*)})$ for some $m^* \in Z_p \setminus \{-\gamma\}$. First, $B^{\ast\cdot}$ picks $i$ from $[q_v + 1]$ uniformly at random, and:

- If $i \leq i' \leq q_v$, $B^{\ast\cdot}$ takes $i^*$ as the guess for the first time that a fresh and valid $(m^*, A^*)$ pair appears for Check.
- Otherwise (i.e., $i^* = q_v + 1$), $B^{\ast\cdot}$ considers the first fresh and valid pair appears in the output of $B$.

Given $\{(m_i, A_i)\}_{i=1}^n$, we define $f(x) = \prod_{i=1}^n (x + m_i) = \prod_{n-j
ot=1}^n (x + m_j) = \prod_{n-j
ot=1}^n (\gamma_j + m_j)$ for each $i \in [n]$. Then, using techniques by [16], $B^{\ast\cdot}$ can compute $g' = \prod_{n-j
ot=1}^n (g^\gamma)^{\gamma_j} = g^{\gamma/(\gamma + m^*)}$, $w = \prod_{n-j
ot=1}^n (g^\gamma)^{\gamma_j-1} = g^{\gamma/(\gamma + m^*)}$ and $A_i = \prod_{n-j
ot=1}^n (g^{\gamma_j})^{\gamma_j} = g^{\gamma/(\gamma + m^*)}$ for each $i \in [n]$. Next, $B^{\ast\cdot}$ returns $g', w$ and $(A_i, A_i)^{n-1}_{i=1}$ to $B$ and responds the i-th Check($\gamma$, $m_i^*$, $A_i^*$) query for $B$ as follows:

- If $i = i^*$, $B^{\ast\cdot}$ sets $(m, A) = (m_i^*, A_i^*)$ and aborts.
- Otherwise, $B^{\ast\cdot}$ returns 1 if $(m_i, A_i) = (m_i^*, A_i^*)$ for some $1 \leq i \leq n$ and 0 otherwise.

If $B^{\ast\cdot}$ does not abort, $B$ outputs $(m^*, A^*)$ and $B^{\ast\cdot}$ sets $(m, A) = (m^*, A^*)$.

If $B^{\ast\cdot}$ guesses correctly with probability $1/(q_v + 1)$, its simulation is perfect, and the tuple $(m, A)$ is fresh and valid. Thus, we have $\gamma \notin \{m_i\}_{i=1}^n$ since for any valid pair $(m, A)$ under $g'$ and $\gamma$, $A$ is uniquely determined by $m$. Let $f(x) = f'(x)(x + m) + \theta$ for some $\theta \in Z_p^{*}$, which is also written as $f'(x) = \sum_{j=0}^{n-1} \delta_j x^j$, we have:

$$A = (g')^{1/(\gamma + m)} = g^{f'/(\gamma + m)} = g^{f'/(\gamma + \theta/(\gamma + m))}$$

Finally, $B^{\ast\cdot}$ computes $g^{1/(\gamma + m)} = (A/g')^{1/\theta}$ with $g^{f'/(\gamma + m)} = \prod_{j=0}^{n-1} (g^\gamma)^{b_j}$, and outputs $(m, g^{1/(\gamma + m)})$ as a solution for the $g$-SDH problem.

By Lemma 1, we can straightforwardly bound the difference between Game 3 and Game 2 using a reduction $B_1$ against the experiment in Fig. 4. Specifically, $B_1$ executes just as in Game 2 and interacts with $A$, with the following exceptions:

- $B_1$ outputs $(\text{uid}_1, \ldots, \text{uid}_n)$ in the experiment in Fig. 4 and then receives $w$ and $\{(\text{uid}_i, A_i)\}_{i=1}^{n-1}$. $B_1$ sets $w$ as $\text{isp}$ and sets $A_1$ as the credential of user $i$.
- For each $\text{VERIFY}(\text{uid}', m, \sigma)$ query, if $\text{uid}' = \text{uid}$, when $A$ is computed with a witness $a$ extracted by Ext, algorithm $B_1$ returns 1 if Check($\gamma$, $\text{uid}$, $A$) = 1.
- When $B_1$ finds a username-credential pair $(\text{uid}^*, A^*)$ such that Check($\gamma$, $\text{uid}^*$, $A^*$) = 1 but $(\text{uid}^*, A^*) \notin \{(\text{uid}_i, A_i)\}_{i=1}^n$, then return 1. Otherwise, return 0.

Fig. 4: Experiment for Lemma 1

Experiment $\text{Exp}_2(B)$

$(m_1, \ldots, m_n, \text{state}) \leftarrow B(1^\lambda)$.

$$\gamma \leftarrow Z_p^*, w \leftarrow g^\gamma; \text{for each } i \in [n], A_i \leftarrow g^{1/(\gamma + m_i)}.$$  

$$(m^*, A^*) \leftarrow B^{\text{Check}(\gamma, \cdot)}(\text{state}, g, w, \{(A_i)_{i=1}^n\})$$

If $A^* = g^{1/(\gamma + m^*)}$ and $(m^*, A^*) \notin \{(m_i, A_i)\}_{i=1}^{n-1}$, then return 1. Otherwise, return 0.

Check($\gamma, m, A$): return 1 if $A = g^{1/(\gamma + m)}$ and 0 otherwise.
If the difference between Game 3 and Game 2 is not negligible, $B_1$ can output a forgery $(uid^*, A^*)$ with non-negligible probability. Thus, we have

$$\text{Adv}_2(A) = \text{Adv}_3(A) + (q_v + 1)\text{Adv}_{\text{DH}}^G(O(t' + n^2t_{\exp}), n + 1).$$

**Complete the proof for Case 1.** If $A$ did not reveal the password-wrapped credential of the target user with username $uid^*$ in $\mathcal{U}$, we can bound the advantage of $A$ in Game 3 using a reduction $B_2$ against the experiment in Fig. 4. Specifically, $B_2$ executes just as in Game 3 and interacts with $A$, with the following exceptions:

- $B_2$ picks $i^* \overset{\$}{\leftarrow} [n]$ as the guess that $uid^* = uid_{i^*}$ where $uid^*$ is output by $A$ in its forgery.
- $B_2$ outputs $\{(uid_i, A_i)\}_{i \in [n], i \neq i^*}$ in the experiment in Fig. 4 and then receives $\{(uid_i, A_i)\}_{i \in [n], i \neq i^*}$ and sets $A_{i^*}$ as the credential of user $i$ for each $i \in [n] \land i \neq i^*$.
- For each Verify$(uid, m, \sigma)$ query, when an element $A \in \mathcal{U}$ is computed with a witness $a$ extracted by Ext for $uid = uid^*$, $B_2$ returns 1 if and only if $\text{Check}(\gamma, uid, A) = 1$.
- When $B_2$ finds a username-credential pair $(uid^*, A^*)$ such that $\text{Check}(\gamma, uid^*, A^*) = 1$ and $uid^* = uid_{i^*}$ in some Verify query, $B_2$ outputs $(uid^*, A^*)$ as its forgery.

If $B_2$ guesses correctly with probability $1/n$, $B_2$ needs not to simulate the password-wrapped credential of user $i^*$, and the simulation of $B_2$ is perfect. Thus, we have:

$$\text{Adv}_3(A) \leq n(q_v + 1)\text{Adv}_{\text{DH}}^G(O(t' + n^2t_{\exp}), n).$$

**Continue the proof for Case 2.** If $A$ has revealed the password-wrapped credential of the target user $i^*$ with username $uid^* \in \mathcal{U}$, we continue the proof as follows:

**Game 4.** This game is the same as Game 3, except for replacing the credential $A_i$ of user $i$ with a random element $R_i$ in $G^*$ for each $i \in [n]$; setting $[R_i]_{pw, i}$ as the user $i$’s password-wrapped credential instead of $[A_i]_{pw, i}$; and for each Verify$(uid, m, \sigma)$ query, when $A$ is computed with a witness $a$ extracted by Ext, return 1 if and only if $A = R_i$.

We can bound the difference between Game 4 and Game 3 using a reduction from the $q$-DDHI assumption. The reduction is done by a standard hybrid argument. For every $j \in [n]$, let Game $(3, j)$ be the same as Game 3 except that setting a random $R_i$ in $G^*$ as the credential of user $i$ for each $i \in [j]$ and using $R_i$ to validate the authentication tokens in all Verify$(uid_{i, 0}, *, *)$ queries with $uid_{i, 0} \in \mathcal{U}$. It is easy to see that Games $(3, 0)$ is the same as Game $3$ and Game $(3, n)$ is the same as Game $4$. If adversary $A$ behaves differently between Game $(3, j - 1)$ and Game $(3, j)$ for some $j \in [n]$, we can construct an algorithm $B_3$ breaking the $q$-DDHI assumption. For the reduction, we use the following lemma.

**Lemma 2.** For all adversaries $B$ running in time $t'$, $B$ aims to win in the experiment as described in Fig. 5 (i.e., $\text{Exp}_B$ outputs 1). Then, the $B'$s advantage $\text{Adv}_{\text{DDHI}}^G(t' + n^2t_{\exp}), n + 1$.

**Proof for Lemma 2.** If there exists an adversary $B$ that makes $\text{Exp}_B$ return 1 with probability $\epsilon$, we can construct an algorithm $B_{\text{DDHI}}^G$ that breaks the $q$-DDHI assumption with probability $\epsilon - n/p$ by interacting with $B$ as follows:

- Given a $q$-DDHI instance $(g, g^{\alpha}, ..., g^{\alpha}, T) \in (G^*)^{n+2}$ for some unknown $\alpha \in \mathbb{Z}_p$, $B_{\text{DDHI}}^G$ aims to distinguish $T = g^{t_{\alpha}}$ with a random $T \in G$. Given $\{m_1, ..., m_n, m^*, \text{state}\}$, $B_{\text{DDHI}}^G$ first computes the tuple $(g, g^\gamma, ..., g^{\gamma})$ by the Binomial Theorem where $\gamma = \alpha - m^*$. Then, it computes $g' = g^\gamma$ and $A_i = (g')^{\alpha/m_i}$ for each $i \in [n]$ as in the proof for Lemma 1, and returns $g', w$ and $\{A_i\}_{i=1}^{n}$ to $B$. Next:
  - If $m^*$ is equal to one of $\{m_i\}_{i=1}^{n}$, $B_{\text{DDHI}}^G$ aborts. We also denote this event as abort.
  - Otherwise, $B_{\text{DDHI}}^G$ computes $g'(\gamma) = \prod_{i=1}^{n} (g^{\gamma})^{\delta_i}$, and returns $\sigma = g'(\gamma)\gamma T^\gamma$ to $B$ as the challenge $A_\gamma$, where $f(x) = q(x)(x + m) + \theta$ for some $\theta \neq 0$ and $q(x) = \sum_{n=0}^{n=1} \delta_i x^i$.

Finally, $B_{\text{DDHI}}^G$ returns the output of $B$ as the solution of $q$-DDHI problem. If $T = g^{\alpha/(\gamma)}$, then $\sigma = g'(\gamma)/\gamma T^\gamma = T$, $B_{\text{DDHI}}^G$ returns 1. The probability of abort is $\Pr[\text{abort}] \leq n/p$. Thus, we have the bound claimed by Lemma 2.

By Lemma 2, we can straightforwardly bound the difference between Game 4 and Game 3 using a reduction $B_3$ against the experiment in Fig. 5. In particular, $B_3$ executes just as in Game 3 and interacts with $A$, with the following exceptions:

- $B_3$ outputs $\{(uid_i, A_i)\}_{i \in [n] \setminus [j]}$ and the challenge message $uid_j$ before the setup phase, and receives $\{(uid_i, A_i)\}_{i \in [n] \setminus [j]}$ and sets $A_i$ as the credential of user $i$ for $i \in [n] \setminus [j]$. $B_3$ also receives a challenge credential $X$ on message $uid_j$. Besides, $B_3$ receives $w$ and sets $w$ as $wp$.
- $B_3$ sets $X$ as the credential of user $j$ and uses $X$ to validate the tokens in all Verify$(uid_{j, *}, *, *)$ queries with the extracted $uid_{j, *}$.
- $B_3$ picks $R_i \overset{\$}{\leftarrow} G^*$ and sets $R_i$ as the credential of user $i$ for each $i \in [j - 1]$. $B_3$ uses $R_i$ to validate the tokens in all Verify$(uid_{i, *}, *, *)$ queries with the extracted $uid_i$ for $i \in [j - 1]$.

If $X$ is an authentication tag on message $uid_j$, $B_3$ behaves exactly as in Game $(3, j - 1)$. If $X$ is a random element in $G^*$, $B_3$ behaves exactly as in Game $(3, j)$. Thus, we have

$$\text{Adv}_3(A) = \text{Adv}_4(A) + n\text{Adv}_{\text{DH}}^G(O(t' + n^2t_{\exp}), n).$$

In Game 4, the credentials of all users are uniformly random in $G^*$. Since $H_{\mathcal{G}}$ is a random oracle, $[R_i]_{pw, i} = R_i \cdot H_{\mathcal{G}}(pw_i)$ is random in group $G$ for every $i \in [n]$. Thus, the only way for getting a credential $R_i$ is to mount online dictionary attacks via Verify for each $i \in [n]$ when the password of user $i$ is not revealed. From the simulation-sound extractability of SPK, we know that $A$ must recover $R_i$ from $[R_i]_{pw, i}$ in order to forge an authentication token on (extracted) username $uid^* = uid_{i^*}$ for any $i \in [n]$. Therefore, we obtain $\text{Adv}_3(A) \leq q_v/|D|$. 

**Fig. 5: Experiment for Lemma 2**
In the final game, we bound the adversary’s advantage by \( q_e / |D| \). This completes the proof.

#### D. Strong Authentication with PBC

Similar to digital signature schemes, \( \pi_{\text{PBC}} \) (as well as other PBC schemes) can be used as a strong authentication mechanism as follows:

1) The server chooses a challenge message \( m \) at random, and sends it to the user.

2) The user generates an authentication token \( \sigma \leftarrow \text{Sign}(\text{uid}, \text{pw}, \text{cre}, \gamma) \), and sends \((\text{uid}, \sigma)\) to the server.

3) Upon receiving \((\text{uid}, \sigma)\), the server verifies the user by checking whether \( \text{Verify}(\gamma, \text{uid}, m, \sigma) = 1 \).

A PBC authenticator can be implemented entirely with software. The user authenticates to server through “something possessed” (i.e., the PBC authenticator) and “something known” (i.e., the password). It provides superior security guarantees than the other software authenticators in the setting that the authenticator is leaked by device stolen/broken, as the latter are vulnerable to off-line attacks. However in the setting where the attacker installs malware (e.g., key-logger) on the user’s device and obtains both the password and the credentials, it might not protect the user and additional mitigation is needed to prevent such key-logger attacks. One possible mitigation is to use an anti-malware software such as IBM’s Trusteer Rapport [7], which is available with PC, android and iOS.

In practice, PBC authenticators can be implemented by the operating system or browser via providing APIs which can be called by applications, or also be implemented by mobile applications or browser extensions using common cryptographic libraries alone.

Compared with traditional authenticator for strong authentication, PBC authenticator does not become another thing to be remembered to carry, as it can be implemented across different devices (e.g., mobile phones and desktop computers) simultaneously to perform strong authentication. Moreover, the PBC authenticator can be stored on a cloud server for backup, which enables the user to fetch it on a new device for authentication in the case that the old device carrying the authenticator is compromised or lost. By contrast, losing an authenticator for hardware-based mechanisms [45], [57], [60] means losing all credentials bound to it. The W3C’s web authentication specification [60] suggests to register multiple credentials for the same user as backup for authenticator lost.

In PBC, a user encrypts and decrypts its credentials with passwords locally, which means that it can change passwords in an off-line manner, by decrypting the wrapped credentials with old passwords and then encrypting them with new ones. Note that the server does not store a password-related file as usual, the system also provides security against offline attacks in the event of server compromise (assuming that the attacker does not also compromise the password-wrapped credentials).

For practical security, we suggest that the server should protect \( \gamma \) with a secure hardware, and should implement the Issue algorithm with a protection from being utilized as an oracle which enables an attacker to query credentials belonging to honest users. Such protection can be implemented with some common measures such as out-of-band devices. To avoid online attacks, both the rate-limiting mechanisms in [31] that effectively limits the number of failed authentication attempts that can be made on the user’s account, and the related techniques [31] reducing the likelihood that an attacker locks legitimate users out by abusing the mechanism, are applicable to PBC-based authentication systems.

![Fig. 6: The application of PBC for holder-of-key assertion.](image)

### V. Application of PBC in Federated Identity

In this section, we show the applications of PBC schemes in federated identity systems, where \( \Pi_{\text{PBC}} \) is taken as an example. We first present a trivial application of using PBC for the authentication between user and IdP, and then describe how PBC can be applied for holder-of-key assertions with a privacy-preserving option. In the second application, we provide a technique to transform the designated-verifiability of PBC scheme to publicly-verifiability with the help of an IdP, such that the user can prove the possession of its credential to any RP. Furthermore, to meet the requirement of privacy-preserving scenarios, such as the user login at RPs with pseudonyms allocated by the IdP, our application also supports an option that the user prove the possession of key for the holder-of-key assertion in a privacy-preserving way.

For simplicity, our description only presents the details that are related to the application of PBC, and can be adopted by various federation systems implementing either modes, such as OpenID Connect [51], OAuth 2.0 [33], [36] and SAML2 [46], [59]. Based on a PBC scheme \( \Pi_{\text{PBC}} = (\text{Setup}, \text{KeyGen}, \text{Issue}, \text{Sign}, \text{Verify}) \), the applications are described as follows:

#### A. Application for user-IdP authentication

**Setup.** The IdP chooses the public parameters \( pp \) from a publicly-trusted source such as a standard, generates the secret key \( \gamma \) and public parameter \( w \) via executing KeyGen. Then it publishes \( pp \) and \( w \).
Registration. The user interacts with the IdP via executing the interactive protocol issue\( (sk, Reg) \Rightarrow (uid, pw) \), and stores the password-wrapped credential \([cre]_{pw}\) to a preferred storage.

Authentication. In common federated identity systems, the user first visits the RP, and is then redirected to the IdP for authentication.

1) The IdP generates a challenge \( n_{I} \leftarrow \{0, 1\}^{\lambda} \), and sends \( n_{I} \) to the user as a challenge.
2) Upon receiving \( n_{I} \), the user generates the authentication token \( \sigma_{I} = (T, \pi_{T}) \) via \( \text{Sign}(uid, pw, [cre]_{pw}, n_{I}) \), and sends \( (uid, \sigma_{I}) \) to the IdP.
3) The IdP verifies the user’s authentication token \( \sigma_{I} \) by checking \( \text{Verify}(\gamma, uid, n_{I}, \sigma_{I}) = 1 \), and continues the protocol flow of identity federation if the check passes.

B. Application for holder-of-key assertion

The authentication token \( \sigma = (T, \pi_{T}) \) of PBC presented in section IV-B is a signature proof of knowledge:
\[
\pi_{T} \leftarrow \text{SPK}'\{(a) : g^{a} = PK\} (m) \text{ for } PK = T^{\gamma + uid},
\]
which can only be checked by the designated verifier (i.e., IdP) with the knowledge of \( \gamma \). For a verifier who does not hold \( \gamma \) (e.g., RP), it needs the value of \( PK \) and the knowledge that \( PK \) is generated correctly in the form \( T^{\gamma + uid} \) for an unknown \( \gamma \). Thus, we let IdP provide \( PK \) with a proof of knowledge that proves the validity of \( PK \).

In this application, the Setup and Registration phases are the same as in the application presented in Section IV-A. We present the Authentication phase as follows (also see Fig. 6):

1) The user authenticates to the IdP first, where an ephemeral private-key \( \alpha \leftarrow Z_{p}^{*} \) is chosen and used in the \( \text{Sign} \) algorithm. \( \alpha \) is kept for further usage.
2) If IdP authenticates the user successfully with an authentication token \( \sigma_{I} = (T, \pi_{T}) \), it generates a holder-of-key assertion. In this step, we provide two options as follows:

a) (The privacy-preserving option.) When the privacy of the user is required to be protected, the IdP calculates \( PK = T^{\gamma} \), puts \( (T, PK) \) in assertion, and signs assertion with a proof of knowledge that \( \log_{T}(PK) \) is equal to that of \( \gamma \):
\[
\pi_{PK} = \text{SPK}'\{(\gamma) : w = g^{\gamma} \land PK = T^{\gamma}\} (\cdot).
\]

b) Otherwise, the IdP calculates \( PK = T^{\gamma + uid} \), and puts \( (T, PK) \) in assertion. Then, it signs assertion with a signature proof of knowledge that the discrete logarithm of \( \log_{T}(T^{\text{uid}} \cdot PK) \) is equal to \( \gamma \):
\[
\pi_{PK} = \text{SPK}'\{(\gamma) : w = g^{\gamma} \land T^{\text{uid}} \cdot PK = T^{\gamma}\} (\cdot).
\]

Next, assertion and \( \pi_{PK} \) (or \( \pi_{PK} \)) are presented to the RP via either the front-channel or back-channel modes. In the front-channel mode, assertion and \( \pi_{PK} \) (or \( \pi_{PK} \)) are transferred to the RP via the user. As for the back-channel mode, only a reference to assertion (e.g., an authorization code) is transmitted to the RP via user. Then, the RP redeems assertion and \( \pi_{PK} \) by sending the reference to the IdP. Upon receiving the reference and checks its validity, the IdP responds with assertion and \( \pi_{PK} \) (or \( \pi_{PK} \)).
3) Upon receiving assertion and \( \pi_{PK} \) (or \( \pi_{PK} \)), the RP requests the user to prove the possession of key corresponding to \( PK \) (or \( PK \)) by sending a random challenge \( n_{R} \leftarrow \{0, 1\}^{\lambda} \).
4) The user generates a proof-of-possibility of the ephemeral private-key \( \alpha \) w.r.t. to the ephemeral public-key \( PK \), which also includes two options:

a) (The privacy-preserving option.) For this option, it calculates a privacy-preserving authentication token:
\[
\pi_{R} \leftarrow \text{SPK}''\{(a, uid) : T^{-uid} \cdot g^{a} = PK\}(n_{R}),
\]
and sends \( \pi_{R} \) to the RP.

b) Otherwise, it calculates:
\[
\pi_{R} \leftarrow \text{SPK}''\{(a) : g^{a} = PK\}(n_{R}),
\]
and sends \( (uid, \pi_{R}) \) to the RP.
5) The RP checks the validity of assertion as well as the user’s proof-of-possibility of key as follows:

a) (The privacy-preserving option.) For this option, the RP first checks the validity of \( \pi_{PK} \) with \( w, T \) and \( PK \), then checks \( \pi_{R} \) with \( T \) and \( PK \), and accepts if all checks pass.

b) Otherwise, the RP checks \( \pi_{PK} \), and use \( PK \) and \( uid \) to verify \( \pi_{R} \).

Instantiation of \( \text{SPK}' \). Here, based on the Chaum-Pedersen protocol [26] and Fiat-Shamir heuristic [28], \( \text{SPK}' \) can be instantiated as follows for a cryptographic hash function \( H' : \{0, 1\}^{*} \rightarrow Z_{p} \):

Prove: The algorithm picks \( r \leftarrow Z_{p} \) and computes \( R_{1} \leftarrow T^{r} \) and \( R_{2} \leftarrow g^{r} \). Then, it computes \( s \leftarrow r + c \cdot \gamma \mod p \), where \( c \leftarrow H'(g, T, w, T^{-uid} \cdot PK, R_{1}, R_{2}) \). Finally, it outputs a proof \( \pi_{PK} \leftarrow (c, s) \).

Verify: Given a tuple \( (g, T, w, PK) \), a proof \( \pi_{PK} = (c, s) \), compute \( R_{1}' \leftarrow T^{s} \cdot (T^{-uid} \cdot PK)^{-c} \) and \( R_{2}' \leftarrow g^{s} \cdot w^{-c} \), and then calculate \( c' \leftarrow H'(g, T, w, T^{-uid} \cdot PK, R_{1}', R_{2}') \). Output 1 if \( c' = c \) and 0 otherwise.

Instantiation of \( \text{SPK}'' \). \( \text{SPK}'' \) can also be instantiated with a cryptographic hash function \( H'' : \{0, 1\}^{*} \rightarrow Z_{p} \):

Prove: Pick \( r_{1}, r_{2} \leftarrow Z_{p} \) and compute \( R \leftarrow T^{-r_{1}} \cdot g^{r_{2}} \). Then, compute \( c \leftarrow H''(g, T, R, m) \), \( s_{1} \leftarrow r_{1} + c \cdot uid \mod p \) and \( s_{2} \leftarrow r_{2} + c \cdot a \mod p \). Finally, output \( \pi_{PK} \leftarrow (c, s_{1}, s_{2}) \).

Verify: Given a tuple \( (g, T, \gamma) \) and proof \( \pi_{PK} = (c, s_{1}, s_{2}) \), compute \( R' \leftarrow T^{-s_{1} + c \cdot \gamma} \mod p \cdot g^{s_{2}} \) and calculate \( c' \leftarrow H''(g, T, R', m) \). Finally, output 1 if \( c' = c \) and 0 otherwise.

As we stated in Section IV-B, \( \text{SPK} \) could be instantiated with standardized signature algorithms (e.g., ISO/IEC 14888-3 [5]), where \( \alpha \) is used as the private key and \( PK \) is used as the public key, which means that the user could prove its possession of the key \( \alpha \) with \( RP \) in various manners. Furthermore, the application of PBC in federated identity systems does not require the user to store \( \alpha \), since each \( \alpha \) is only used in one session.

We note that this application does not impact the security of PBC against offline attack, even if the authentication tokens
have been converted to be publicly-verifiable. Each time when
the user tries to perform the transform for an authentication
token \( \sigma \), it must authenticate to the IdP. Thus, the request from
attacker can be detected and blocked.

VI. PERFORMANCE EVALUATION

In this section, we evaluate the performances of the primitive proposed in Section V-B as a strong authentication mechanism, as well as its application for holder-of-key assertions, as described in Section V-C. Our implementations take place at the security level of 128 bits, where the secp256r1 \([1]\) curve is adopted as the underlying public parameter \( pp \). The hash algorithms (i.e., \( H, H', H'' \), and \( H_0 \)) are instantiated by SHA-256 with different prefixes.

A. Implementations

**Strong Authentication Mechanism.** To simulate the practical scenario where PBC is adopted for strong authentication over the Internet, we implement \( \text{H}_\text{PBC} \) in a multi-platform manner. In particular, we develop the Sign algorithm in JavaScript as an implementation of PBC software authenticator, where sjcl 1.0.2 is used as the underlying cryptographic library. This corresponds to a wide range of application scenarios where the user authenticates to the server through web and mobile platforms. The Verify algorithm is developed in Java, which corresponds to the practical scenario that the server serves its clients via the Java-based website frameworks (e.g., the Spring Framework), and uses Bouncy Castle 1.60 as the underlying library of cryptographic primitives.

To be more specific, the implementation of Sign algorithm is encapsulated in a browser extension for Chrome, and the Verify algorithm is integrated into a demo website. We note that this is not the only way to deploy the PBC based strong authentication mechanism in practice. The Sign algorithm could also be implemented as application for mobile platform or be integrated into the operation systems and browsers as a compliant authenticator, and the Verify algorithm could be implemented in various programming languages such as Python and Go, and employed by servers of different architectures. In a concrete authentication process, the user first visits the demo server with its browser, and receives a challenge message embedded in a HTTP page. Then, it uses the browser extension to read the challenge from the page, and calculates the authentication token by providing its username, password and password-wrapped credential to the extension. Next, the extension writes the authentication token as an element at the page, which triggers the webpage to send the token and the user’s username back to the server. Finally, the server verifies the token via the Verify algorithm.

**Holder-of-Key Assertion Mechanism.** The application of PBC in federated identity systems (see Section V) could be divided into three parts including:

1) The user authenticates to the IdP
2) The IdP issues a holder-of-key assertion and signs it with \( \text{SPK}' \), and the RP verifies the signature over assertion by executing \( \text{Verify}_{\text{SPK}'} \). We denote this part as \( \text{HoKA} \).
3) The user proves its possession of secret key \( a \) to the RP. We denote this part as \( \text{PoPK} \).

Since the implementation of the first part have been developed as a strong authentication mechanism, the implementation of holder-of-key mechanism consists of two parts:

**HoKA Part.** We develop an “RP” and an “IdP” as two Java applications which could interact with each other. During the interaction, the “IdP” calculates \( PK \) for a given \( T \), issues an assertion including \( PK \) and \( T \) with the format defined in SAML2 \([46]\), signs it with \( \text{SPK}' \), and sends the assertion and signature to the “RP”. Upon receiving the assertion and signature, the “RP” obtains \( PK \) and \( T \), and verifies the signature with \( \text{Verify}_{\text{SPK}'} \).

**PoPK Part.** The user end operation is also performed with a JavaScript implementation as in the case of strong authentication, and interacts with a prototype of “RP” which performs the verification.

B. Performance Evaluation and Comparison with ECDSA

The performances of the PBC based schemes are shown in Table [1], where \( \text{AUTH} \) denotes the strong authentication mechanism, \( \text{HoKA} \) denotes the process of signing and verifying of Holder-of-Key Assertions, and \( \text{PoPK} \) denotes the Proof-of-Possession of Key between the user and RP. We also use \( \text{PoPK-PBC} \) and \( \text{HoKA-PBC} \) to refer to the privacy-preserving option for the holder-of-key assertion mechanism. For comparison, we also develop the strong authentication mechanism and holder-of-key assertion mechanism with ECDSA. To be more specific, we implement the user end computation of strong authentication and proof-of-possession of key with ECDSA by employing a tamper-resistant hardware module embedded in a USB device, that is, the Atmel AT88CK590 evaluation kit. The kit is equipped with an Atmel ECC508A chip which could perform ECDSA signing operations and protect the private keys. We write a Chrome application in JavaScript to interact with the device. For the holder-of-key assertion mechanism with ECDSA, we implement the corresponding “IdP” and “RP” in the same way as PBC, where the “IdP” signs an assertion with ECDSA and the “RP” verifies the signature. We do not employ the hardware module in the latter implementation since it does not include user end operation.

Column 1 and 2 show the time breakdown by token/assertion generation and token/assertion verification.

- For \( \text{AUTH-PBC/ECDSA} \), token generation refers to the calculation of authentication token, and token verification refers to checking of an authentication token.
- For \( \text{HoKA-PBC/ECDSA} \), assertion generation refers to the process of calculating \( PK \) and signing the assertion, and assertion verification refer to the validation of the signature over assertion.
- For \( \text{PoPK-PBC/ECDSA} \), token generation refers to the signing of \( n_H \) with the ephemeral private key \( a \), and token verification refers to the verification of the signature with \( PK \).

The results for \( \text{AUTH-PBC/ECDSA} \) are measured with the Chrome developer tool and Java nanotime function, and are the average of 100 runs. The comparison result shows that the time cost for verifying PBC tokens is almost the same as verifying ECDSA signatures, but the generation of
PBC token is faster than signing with ECDSA for about 100ms. The result for HoKA-PBC/PBC/ECDSA are also measured with the average of 100 runs, which shows that the PBC based mechanism is slightly slower than the mechanism with ECDSA, with difference limited in 2ms. For PoPK- PBC/ECDSA, the advantage of PBC based scheme is more obvious, since the user end implementation only signs the challenge with \( \alpha \) and does not have to repeat the process of decryption and randomizing the credential.

Column 3-7 of Table I present the average time costs when being tested over Local Area Network (LAN) and Wide Area Network (WAN), where each result takes the average of 10 runs. In the LAN setting, the implementations are connected via an 1Gbps network with ping time less than 1 ms. In the WAN setting for AUTH/PoPK-PBC/PBC/ECDSA, we employ several VPN servers to “fix” the latency between the implementations, where all the network latencies are measured with the ping command. We note that, due to the instability of Internet, the latency can not be truly fixed to a constant value. In our experiment, each result is measured with relatively stable latency within \( \pm 5\) ms. In the WAN setting for HoKA-PBC/PBC/ECDSA, we adopt Linux tc command to control the network latency. The results for AUTH/PoPK- PBC/ECDSA are measured via Chrome developer tool from when the user begins to calculate the token, to when the browser receives the authentication success response from server or “RP”. It shows that the strong authentication (resp., proof-of-possession of key) with PBC obtains better efficiency than with ECDSA while being used on web by saving 26%-36% (resp., 41%-55%) time, since the user end calculation could be performed by software. When the privacy-preserving option is used, the proof-of-possession of key process saves 30%-38% time. The results for HoKA-PBC/PBC/ECDSA are measured from when the “IdP” begins to issue the assertion (i.e., calculates \( PK \)), to when it receives a response from the “RP”, which means the check has completed. In this scenario, the PBC based scheme is a bit slower than ECDSA. However, it is still acceptable since it only takes more time from 1.03x to 1.10x, and the slowdown is relatively small compared to the latency raised by network and user end computation.

Experiment Environment. The results in Table I are obtained with a workstation and a desktop computer. For AUTH/PoPK- ECDSA/PBC, the user end implementation is run on a desktop computer, which has an Intel Core i5-3470 processor with 4 cores running at 3.2 Ghz each. The underlying OS is Windows 10 Enterprise Edition. The implementation of server (or “RP”) is run by a Lenovo X3650 workstation, which has an Intel Xeon E5-2640 v3 processor with 8 cores running at 2.6 Ghz each, and runs the operating system of CentOS 7. For HoKA- ECDSA/PBC, both “IdP” and “RP” are run by the workstation as applications which are run by different cores.

<table>
<thead>
<tr>
<th></th>
<th>generation</th>
<th>verification</th>
<th>LAN</th>
<th>WAN</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>30ms</td>
<td>60ms</td>
<td>90ms</td>
<td>120ms</td>
</tr>
<tr>
<td>AUTH-ECDSA</td>
<td>272.41*</td>
<td>1.1</td>
<td>300.11*</td>
<td>342.41*</td>
</tr>
<tr>
<td>AUTH-PBC</td>
<td>187.51</td>
<td>1.0</td>
<td>192.41</td>
<td>224.91</td>
</tr>
<tr>
<td>PoPK-ECDSA</td>
<td>271.11*</td>
<td>1.1</td>
<td>305.41</td>
<td>334.31*</td>
</tr>
<tr>
<td>PoPK-PBC</td>
<td>100.61</td>
<td>1.0</td>
<td>125.01</td>
<td>149.71</td>
</tr>
<tr>
<td>PoPK-PBC’</td>
<td>167.31</td>
<td>1.0</td>
<td>190.51</td>
<td>223.71</td>
</tr>
<tr>
<td>HoKA-ECDSA</td>
<td>0.7</td>
<td>1.0</td>
<td>3.3</td>
<td>34.7</td>
</tr>
<tr>
<td>HoKA-PBC</td>
<td>2.1</td>
<td>2.4</td>
<td>5.1</td>
<td>38.3</td>
</tr>
<tr>
<td>HoKA-PBC’</td>
<td>2.0</td>
<td>1.9</td>
<td>5.0</td>
<td>37.2</td>
</tr>
</tbody>
</table>

TABLE I: The comparison between the runtimes (in milliseconds) of PBC and ECDSA when being used as strong authentication mechanisms or applied in federated identity systems for LAN and WAN. In the cases where the results are calculated with the average of several runs, their standard derivations are presented in brackets. \( \dagger \) denotes that the algorithm or user end computation is implemented by JavaScript. \( \ast \) denotes that this implementation employs a tamper-resistant hardware module at user end. By PoPK-PBC’ and HoKA-PBC’, we refer to the privacy-preserving option of PBC-based holder-of-key mechanism.

VII. CONCLUSION

In this paper, we propose a strong authentication mechanism which does not rely on tamper-resistant hardware modules at user end, from a new primitive called Password-Based Credential (PBC), and show its application in federated identity systems. We also present \( \Pi_{\text{c}} \) as an efficient construction of PBC, and evaluates its performances. The evaluation result shows that the schemes with PBC also obtain better efficiency than the traditional strong authentication mechanisms with tamper-resistant hardware modules. As future work, we plan to develop our PBC based schemes in real-world application scenarios. It can be expected that our schemes help organizations and enterprises to provide their users with friendly and strong authentication experiences in the future.

ACKNOWLEDGMENT

The authors would like to thank the anonymous reviewers of NDSS 2020, and Jeremiah Blocki for their helpful comments and suggestions. This work is supported by the National Key Research and Development Program of China (No. 2017YFB0802500, 2017YFB0802000) and the National Natural Science Foundation of China (No. 61728208, U1536205, 61802376 and 61802021).

REFERENCES

“Lost electronic devices can lead to data breaches.”
https://www.azcentral.com/story/money/business/tech/2015/09/30/


“Creating a SAML holder-of-key token using the API”
webSphere.nd.multiplatform.doc/actwbss_createholderofkeytoken.html
March 2019.

“BM Trusteer Rapport - Helps financial institutions detect and prevent malware infections and phishing attacks, maximizing protection for their customers.”
ware-protection December 2019.

“Sign in with Apple - the fast, easy way to sign in to apps and websites.”

PAKE password-authenticated key exchange protocol,” in 2015 IEEE
Symposium on Security and Privacy, SP 2015, San Jose, CA, USA, May
17-21, 2015, pp. 571–587.

D. Baghdasaryan, R. Sasson, B. Hill, J. Hodges, and K. Yang, “FIDO

A. Barki, S. Brunet, N. Desmoullins, and J. Traoré, “Improved alge-
braic macs and practical key-verification anonymous credentials,” in

M. Bellare, D. Pointcheval, and P. Rogaway, “Authenticated key ex-
change secure against dictionary attacks,” in Advances in Cryptology -
EUROCRYPT 2000, International Conference on the Theory and
Application of Cryptographic Techniques, Bruges, Belgium, May 14-18,

M. Bellare and P. Rogaway, “Random oracles are practical: A paradigm
for designing efficient protocols,” in Proceedings of the 1st ACM Con-
ference on Computer and Communications Security, CCS ’93, Fairfax,
Virginia, USA, November 3-5, 1993 , pp. 62–73.

“The AuthA Protocol for PasswordBased Authenticated Key

D. Boneh and X. Boyen, “Secure identity based encryption without ran-
don oracles,” in Advances in Cryptology - CRYPTO 2004, 24th Annual
International Cryptology Conference, Santa Barbara, California, USA,

——, “Short signatures without random oracles,” in Advances in Crypt-
tology - EUROCRYPT 2004, International Conference on the Theory
and Applications of Cryptographic Techniques, Interlaken, Switzerland,

J. Bonneau, C. Herley, P. C. van Oorschot, and F. Stajano, “The quest
to replace passwords: A framework for comparative evaluation of web
authentication schemes,” in IEEE Symposium on Security and Privacy,
S&P 2012, 21-23 May 2012, San Francisco, California, USA, pp. 553–
567.

H. Brekalo, R. Strackx, and F. Piessens, “Mitigating password database
breaches with intel SGX,” in Proceedings of the 1st Workshop on
Software for Trusted Execution, SystEX@Middleware 2016, Trento,
Italy, December 12, 2016, pp. 1-1-1.

J. Camenisch, M. Drijvers, and M. Dubovitskaya, “Practical uc-
secure delegatable credentials with attributes and their application to
blockchain,” in Proceedings of the 2017 ACM SIGSAC Conference on

J. Camenisch, A. Lehmann, G. Neven, and K. Samelin, “Virtual smart
cards: How to sign with a password and a server,” in Security and
Crypto
graphy for Networks - 10th International Conference, SCN 2016,

J. Camenisch and M. Stadler, “Efficient group signature schemes for
large groups (extended abstract),” in Advances in Cryptology - CRYPTO
‘97, 17th Annual International Cryptology Conference, Santa Barbara,

P. Chaidos and G. Couteau, “Efficient designated-verifier non-interactive
zero-knowledge proofs of knowledge,” in Advances in Cryptology -
EUROCRYPT 2018 - 37th Annual International Conference on the
Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel,
2018, pp. 193–221.

Cryptology Conference, Santa Barbara, California, USA, August 20-

M. Chase, S. Meiklejohn, and G. Zaverucha, “Algebraic macs and
key-verification anonymous credentials,” in Proceedings of the 2014
ACM SIGSAC Conference on Computer and Communications Security,

D. Chaum and H. V. Antwurpen, “Undeniable signatures,” in Advances
in Cryptology - CRYPTO ’89, 9th Annual International Cryptology
Conference, Santa Barbara, California, USA, August 20-24, 1989,

D. Chaum and T. P. Pedersen, “Wallet databases with observers,” in
Advances in Cryptology - CRYPTO ’92, 12th Annual International
Cryptology Conference, Santa Barbara, California, USA August 16-20,

A. Everspaugh, R. Chatterjee, S. Scott, A. Juels, and T. Ristenpart, “The
pythia PRF service,” in 24th USENIX Security Symposium, USENIX
Security 2015, Washington, D.C., USA, August 12-14, 2015, pp. 547–
562.

A. Fiat and A. Shamir, “How to prove yourself: Practical solutions
to identification and signature problems,” in Advances in Cryptology -
CRYPTO ’86, Santa Barbara, California, USA, 1986, Proceedings, pp.
186–194.

V. Galindo, R. Lindemann, U. Martini, C. Edwards, and J. Hodges,

secure against adaptive chosen-message attacks,” SIAM J. Comput.,

P. Grassi, M. Garcia, and J. Fenton, “NIST Special Publication 800-
63-3 Digital Identity Guidelines,” National Institute of Standards
and Technology, 2017.

J. Groth, “Simulation-sound NIZK proofs for a practical language
and constant size group signatures,” in Advances in Cryptology -
ASIACRYPT 2006, 12th International Conference on the Theory and
Application of Cryptography and Information Security, Shanghai, China,


M. Jakobsson, K. Sako, and R. Impagliazzo, “Designated verifier proofs
and their applications,” in Advances in Cryptology - EUROCRYPT ’96,
International Conference on the Theory and Application of Cryptographic
143–154.

S. Jarecki, H. Krawczyk, M. Shrivastava, and N. Saxena, “Two-factor
authentication with end-to-end password security,” in Public-Key Crypt-
tography - PKC 2018 - 21st IACR International Conference on Practice
and Theory of Public-Key Cryptography, Rio de Janeiro, Brazil, March

M. Jones, J. Bradley, and H. Tschofenig, “Proof-of-Possession Key
Semantics for JSON Web Tokens (JWTs),” RFC 7800, 2016.

M. Jones, “The Increasing Importance of Proof-of-Possession to the
Web,” https://www.w3.org/2012/webcrypto/webcrypto-next-workshop/

A. J. Menezes, P. C. Van Oorschot, and S. A. Vanstone, Handbook of
applied cryptography. CRC press.

N. Klingenstein and S. Tom, “SAML V2.0 Holder-of-Key Web Browser
SSO Profile Version 1.0,” 2010.

D. Kogan, N. Manohar, and D. Boneh, “Tkkey: Second-factor authen-
tication from secure hash chains,” in Proceedings of the 2017 ACM
SIGSAC Conference on Computer and Communications Security,
CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017, pp.
983–999.

K. Krawczyk, A. Paverd, and N. Asokan, “Protecting password
database using trusted hardware,” in Proceedings of the 1st Workshop
on System Software for Trusted Execution, SystEX®Middleware 2016,

Rebirth of a cryptographic password-hardening service,” in 26th USENIX
Security Symposium, USENIX Security 2017, Vancouver, BC, Canada,


