Textile & Leather Review

ISSN 2623-6281| www.tlr-journal.com | 10.31881/TLR

KSDAuth: A Private-Key-Sharding-Based Distributed Authorization Mechanism for Credential Forgery Mitigation

Mingdong He, Fengzheng Liu, Zhenghao Qian, Bo Li, Xuewu Li, Chuangye Zhao, Gehua Fu, Yifan Hu

How to cite: He M, Liu F, Qian Z, Li B, Li X, Zhao C, et al. KSDAuth: A Private-Key-Sharding-Based Distributed Authorization Mechanism for Credential Forgery Mitigation. Textile & Leather Review. 2026; 9:6000-6024. https://doi.org/10.31881/TLR.2026.6000

How to link: https://doi.org/10.31881/TLR.2026.6000

Published: 5 June 2026

KSDAuth: A Private-Key-Sharding-Based Distributed Authorization Mechanism for Credential Forgery Mitigation

Mingdong He, Fengzheng Liu*, Zhenghao Qian, Bo Li, Xuewu Li, Chuangye Zhao, Gehua Fu, Yifan Hu

Digital Intelligent Operation Center, Guangdong Power Grid Co., Ltd., Guangzhou 510000, Guangdong, China
*13661078204@163.com

Article

https://doi.org/10.31881/TLR.2026.6000

Published 5 June 2026

ABSTRACT

Modern cross-domain access and single sign-on (SSO) mechanisms commonly rely on the signing private key held by an Identity Provider (IdP) to guarantee the authenticity and integrity of issued credentials. Once this signing key is leaked, attackers may forge SAML assertions or JSON Web Tokens (JWTs) that can still pass conventional verification at the Service Provider (SP). To mitigate this single-point key-leakage risk, this paper proposes KSDAuth, a distributed authorization mechanism based on private-key sharding. KSDAuth replaces the conventional single-key signing process with a 3-of-3 collaborative signing process involving the SP, the Authenticator (Auth), and the Token Issuer (Issuer). Each participant holds a distinct private-key share and signs the same canonicalized identity context, which includes user identity, privilege set, issuer, audience, session identifier, request identifier, nonce, and validity period. The SP then reconstructs and verifies the aggregated signature using the original public key. Unlike a drop-in replacement for existing SSO systems, KSDAuth preserves the semantic structure of JWT/SAML identity claims while requiring additional protocol extensions for partial signing, identity-context binding, secure share transmission, and aggregate verification. Under the defined threat model, the security analysis shows that a single compromised key share is insufficient to generate a valid credential, and that inconsistent identity contexts can be detected during aggregate verification. The paper further discusses two-share compromise, replay attempts, original-key leakage, and deployment limitations. The analysis suggests that KSDAuth can reduce credential-forgery risks under clearly stated assumptions, but its practical deployment still depends on secure initialization, authenticated communication channels, SP-side state validation, and further performance and interoperability evaluation.

KEYWORDS

distributed authorization, private key sharding, credential forgery, cross-domain access, single sign-on

INTRODUCTION

The effectiveness of an access-control mechanism depends critically on the trustworthiness and continuity of subject authentication, as well as on its ability to respond to security threats and environmental changes in a timely manner [1]. With the widespread adoption of cloud computing and microservice architectures, Web service login scenarios involving cross-domain access and multiple users, especially those based on SAML and JWT, have become the norm. However, these non-deterministic access-control scenarios are built upon a fragile security premise: the signing private key held by the Identity Provider (IdP) must remain absolutely secure [2].

In recent years, the theft of private keys and other high-value credential-like sensitive information has become an important means for establishing advanced persistent threats (APTs) [3]. Once the underlying private key of the IdP is leaked at a single point, attackers can easily carry out highly destructive credential forgery. As shown by the Golden SAML attack exposed in the SolarWinds supply-chain incident [4–5], as well as signature-forgery threats targeting JWT tokens [6], attackers can use a genuine private key to create apparently legitimate credentials. Such attacks are highly stealthy, capable of bypassing multi-factor authentication, and able to render existing access-control policies ineffective.

Although existing studies have improved authentication reliability in certain environments, for example by implementing dynamically trusted credential validation deep within operating systems [7], these mechanisms usually apply only to relatively deterministic single-host scenarios. Once they are used in non-deterministic environments characterized by frequent cross-domain interactions and many participating entities, they still find it difficult to identify and block access based on perfectly forged credentials. Therefore, how to eliminate credential forgery risks caused by single-node private-key leakage at the architectural level of the authentication mechanism has become an urgent research issue.

Related studies have explored several directions. In response to credential theft and identity spoofing, hybrid identity verification and multi-factor authentication schemes have been proposed [8–10]. Other studies have introduced resilient user-authentication methods based on behavioral or biometric characteristics [11]. In

distributed access-control architecture, decentralized authorization frameworks and data-protection services have also been developed [12–14]. In addition, threshold signature mechanisms based on Lagrange polynomials have provided theoretical inspiration for weakening dependence on the private key of a single node [15]. However, these studies do not directly solve the problem that, once the signing private key of a credential-based cross-domain authentication system is leaked, attackers may still forge credentials that pass conventional verification.

To address this gap, this paper proposes KSDAuth, a distributed authorization technology based on private key sharding. The main idea is to preserve the existing SAML and JWT workflow while reconstructing its trust basis. Instead of allowing a centralized IdP to sign credentials with a complete private key, KSDAuth splits the original private key into multiple shards and assigns them to different roles in the cross-domain access chain. The final authorization result is then determined by collaborative generation and aggregate verification rather than by a single trust center.

The contributions of this paper are as follows.

First, this paper proposes KSDAuth, a distributed authorization mechanism for mitigating credential forgery caused by IdP signing-key leakage. Instead of relying on a single IdP signing key, KSDAuth distributes the signing capability across the SP, Auth, and Issuer.

Second, this paper designs a 3-of-3 private-key sharding mechanism based on Lagrange polynomial interpolation. In the proposed setting, all three participants are required to contribute valid partial signatures before a credential can be accepted. This design improves resistance to single-share leakage, but it also introduces an availability trade-off when one participant is offline.

Third, this paper introduces identity-context binding into the distributed authorization process. User identity, privilege set, issuer, audience, request identifier, session identifier, nonce, and validity period are canonicalized into a single message signed by all participants. This ensures that inconsistent identity information across participants can be detected during aggregate verification.

Fourth, this paper clarifies the deployment impact of KSDAuth on existing JWT/SAML-based SSO systems. KSDAuth preserves the semantic structure of identity claims, but it requires additional middleware or protocol extensions for partial signing, secure token-fragment transmission, and aggregate verification.

Finally, this paper provides a security analysis under explicit adversarial assumptions and discusses the limitations of the current design, including secure setup, two-share compromise, original private-key leakage, node unavailability, and the need for further empirical performance evaluation.

EXPERIMENTAL

Materials and Methods

Basic Concepts of Credential-Based Authentication

Credentials and JWT Tokens

In information technology, credentials refer to information used to verify the identity of a user or system so that access to protected resources can be granted. Such credentials may include passwords, digital certificates, tokens, and keys. In this paper, special attention is given to tokens because token-based identity verification is widely used in Web services.

A token is a digital object containing information about the principal making the request and the access privileges granted to that principal. In typical authentication workflows, credentials are exchanged for tokens to determine permissions. A common example is the JSON Web Token (JWT), which is widely used to securely transmit signed identity information between clients and servers. JWT supports authorization and authentication in a lightweight and stateless manner. It is also representative of the general structure used in many types of identity-related tokens.

A JWT token consists of three parts: Header, Payload, and Signature. As shown in Figure 1, a JWT token includes the Header, Payload, and Signature, which together determine the integrity and authenticity of the token. The Header describes the signing algorithm and token type; the Payload contains identity and authorization claims such as subject, issuer, audience, issue time, and expiration time; and the Signature is generated by signing the encoded Header and Payload with a private key. The Signature is used to ensure the integrity and source authenticity of the token. Since JWT is encoded rather than encrypted, anyone can view the information in the token, but only those with the correct key can verify its authenticity. Therefore, JWT signing mechanisms face security risks similar to those of code-signing systems, especially private-key leakage and signature forgery.

Header: {“alg”: “RS256”, “typ”: “JWT”}

Payload: {"sub": "1234567890", "name": "John Doe", "iss": "KNOXSSO", "aud": "DSX", "iat": 1516239022, "exp": 1516242622}

Figure 1. Basic Composition of a JWT Token

SAML Protocol in Single Sign-On

Single Sign-On (SSO) has been widely adopted as an efficient and secure solution for identity management across multiple services. SAML (Security Assertion Markup Language) is an important open standard in the SSO domain and is used to exchange authentication and authorization data across systems. It defines message formats and protocol bindings that allow an Identity Provider (IdP) and a Service Provider (SP) to exchange user identity information securely.

In digital identity management and access control, the IdP is responsible for verifying users and managing identity information, including issuing tokens or assertions representing verified identity. The SP relies on the IdP to authenticate the user and then grants access to services or resources. A core concept in SAML is the SAML assertion, which is an XML document used to convey authentication, authorization, and attribute information. A typical assertion contains elements such as Subject, Conditions, AuthnStatement, and AttributeStatement. In SSO scenarios, these elements collectively support authentication, authorization, and security protection through digital signatures. An example of a SAML assertion is shown in Figure 2.

```
<saml:Assertion xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion">
  <saml:Subject>
    <saml:NameID Format="urn:oasis:names:tc:SAML:2.0:nameid-format:transient">example_user</saml:NameID>
  </saml:Subject>
  <saml:Conditions NotBefore="2023-08-12T20:00:00Z" NotOnOrAfter="2023-08-12T21:00:00Z"/>
  <saml:AuthnStatement AuthnInstant="2023-08-12T20:00:00Z" SessionIndex="_e45d70a2b09244488975f73783203f01">
    <saml:AuthnContext>
      <saml:AuthnContextClassRef>urn:oasis:names:tc:SAML:2.0:ac:classes:PasswordProtectedTransport</saml:AuthnContextClassRef>
    </saml:AuthnContext>
  </saml:AuthnStatement>
  <saml:AttributeStatement>
    <saml:Attribute FriendlyName="email" Name="urn:oid:0.9.2342.19200300.100.1.3" NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:ur
      <saml:AttributeValue xsi:type="xs:string">user@example.com</saml:AttributeValue>
    </saml:Attribute>
  </saml:AttributeStatement>
  <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
  </ds:Signature>
</saml:Assertion>
```

Figure 2. Example of a SAML Assertion

Cross-Domain Access Model Integrating JWT and SAML

With the widespread use of cloud computing and microservice architectures, modern SSO systems need to support lightweight and efficient cross-domain access. In this context, SAML can provide the overall authentication framework and interoperability, while JWT can serve as the lightweight realization of identity tokens. By using SAML as the framework and JWT as the specific credential form, a secure and efficient cross-domain access model can be built.

In the proposed model, three major roles are involved: the User, the Identity Provider (IdP), and the Service Provider (SP). The IdP is further divided into two functional modules: an Authenticator (Auth), which verifies user identity, and a Token Issuer (Issuer), which issues JWT tokens after successful authentication.

From the token-use perspective, the access-control process can be divided into three stages: token application, token delivery, and token verification. The complete access process can be summarized as follows. First, the User attempts to access the SP application or website. Second, the SP checks whether the User has already logged in and whether a valid JWT token exists. If not, the User is redirected to the IdP login page. Third, the User submits credentials to the IdP, and the IdP verifies them. Fourth, if verification succeeds, the IdP issues a JWT token and returns it to the User through a configured callback URL. Fifth, the User's browser automatically forwards the token to the SP. Finally, the SP verifies the token and, if it is valid, grants access to the requested resource or service. The overall access flow is illustrated in Figure 3.

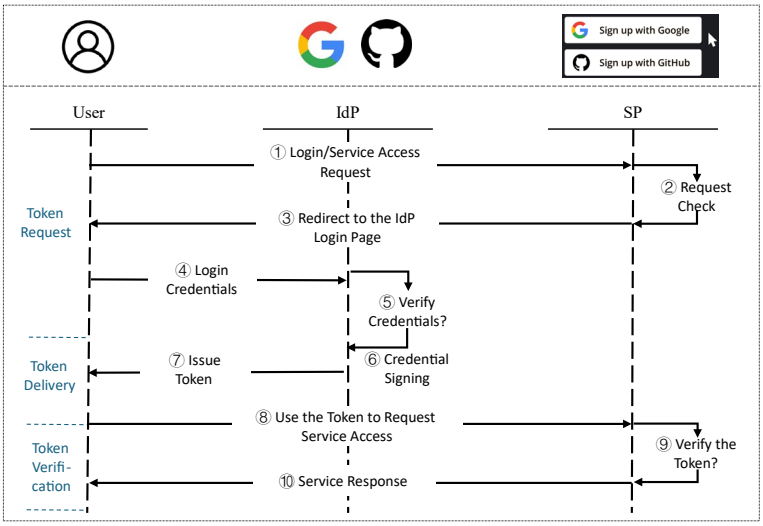

Figure 3. Cross-Domain Access Flow Integrating JWT and SAML

This model combines the SSO features of SAML with the lightweight nature of JWT. It allows users to authenticate once at the IdP and then access multiple SPs conveniently. At the same time, it makes clear that the IdP typically holds the private signing key, while the SP holds the public key used for signature verification. This design simplifies identity federation but also creates a single point of trust around the IdP private key. Once that key is leaked, attackers can forge JWT tokens that are still considered valid under the SP's conventional verification mechanism. The integrated cross-domain access model is shown in Figure 4.

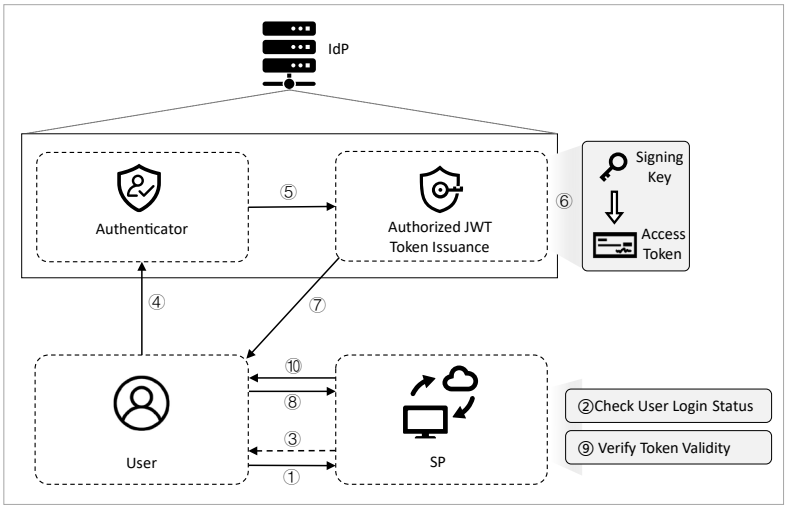

Figure 4. Cross-Domain Access Model Integrating JWT and SAML

Threat Model and Security Assumptions

This paper focuses on credential forgery caused by leakage of signing material in credential-based cross-domain access systems. In a conventional JWT/SAML-based SSO workflow, the IdP or Issuer signs identity credentials using a private signing key, and the SP verifies the received credential using the corresponding public key. If the signing private key is leaked, an attacker may generate forged credentials that appear to be valid under conventional verification.

Adversarial Capabilities

The adversary considered in this paper may attempt to obtain one or more private-key shares from the SP, Auth, or Issuer. The adversary may also construct forged identity attributes, privilege sets, validity periods, and token fragments. In addition, the adversary may attempt to replay previously observed token fragments, mix token fragments from different requests, or submit forged token fragments to the SP. For network

communication, this paper assumes that an adversary may observe public network traffic and may delay, drop, or replay messages that are not protected by authenticated channels. However, communication channels among SP, Auth, and Issuer are assumed to be protected by authenticated and confidential transport mechanisms, such as TLS or mTLS. If these channels are not authenticated and confidential, KSDAuth cannot guarantee the integrity of token-fragment transmission.

Node-Compromise Model

This paper distinguishes three levels of node compromise. First, share leakage means that the adversary obtains the private-key share held by one participant, but does not control that participant's runtime state or session data. Second, runtime compromise means that the adversary controls a participant and may access local request state, cached session data, or identity context information in addition to the private-key share. Third, collusive compromise means that two or more participants are controlled by the adversary and may generate coordinated partial signatures over the same forged identity context. The primary security goal of KSDAuth is to mitigate credential forgery under single-share leakage. The paper also analyzes the security boundary under two-share compromise and full compromise.

Identity Context

In KSDAuth, the identity context is the canonicalized message signed by all participants. It is denoted as:

ctx = H(sub || issuer || audience || sp_id || session_id || request_id || nonce || auth_time || exp || privilege_set)

where sub denotes the user identifier, issuer denotes the credential issuer, audience denotes the target SP, sp_id denotes the service provider, session_id denotes the authenticated session, request_id denotes the current authentication request, nonce provides replay resistance, auth_time denotes the authentication time, exp denotes the expiration time, and privilege_set denotes the authorized privileges. All participants must sign the same ctx. If different participants sign different identity contexts, the reconstructed signature will not verify under the public key.

Communication and Freshness Assumptions

KSDAuth requires that token fragments transmitted among SP, Auth, and Issuer be bound to request_id, nonce, session_id, and timestamp. Before aggregate verification, the SP checks the source of each token fragment, the validity of the request state, the freshness of the nonce, and the expiration time. Token fragments that fail source authentication or freshness checking are rejected before signature aggregation.

Security Goals

The security goals of KSDAuth are as follows:

• Single-share unforgeability: an adversary who obtains only one private-key share cannot generate a complete credential that passes aggregate verification.

• Identity-context consistency: all valid partial signatures must be generated over the same canonicalized identity context.

• Tamper detection: unauthorized modification of identity attributes, privilege sets, or validity periods should cause aggregate verification to fail.

• Replay resistance: token fragments from an old request cannot be reused in a new authentication request.

• Explicit security boundary: KSDAuth does not claim to remain secure when all three participants are fully compromised.

Out-of-Scope Attacks

This paper does not claim to defend against all possible identity-system attacks. If the SP, Auth, and Issuer are all compromised, the adversary may generate valid coordinated partial signatures. If the SP is fully malicious and signs a forged identity context, KSDAuth cannot prevent that SP from participating in the forgery. Attacks against the client browser, the TLS/mTLS infrastructure, the trusted setup environment, or legacy single-signature verification paths are also outside the main scope of this paper.

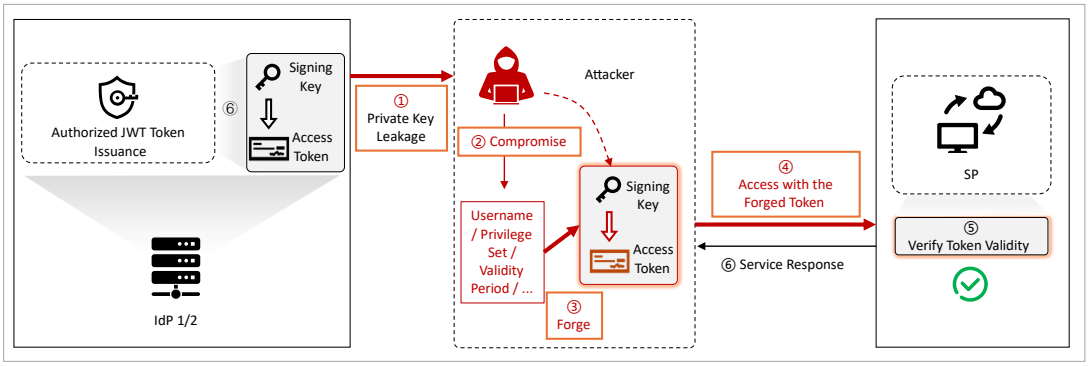

Figure 5. Threat Model: JWT Token Forgery Based on IdP Private Key Leakage

Core Design of KSDAuth

To mitigate credential forgery caused by IdP signing-key leakage, KSDAuth changes the trust basis of conventional credential issuance from single-party signing to multi-party partial signing. The design consists of four core components.

First, KSDAuth preserves the semantic structure of JWT/SAML identity claims, such as subject, issuer, audience, expiration time, and privilege set. However, it does not provide zero-modification compatibility with existing SSO deployments. Instead, it requires additional protocol extensions or middleware support for partial signing, token-fragment transmission, identity-context binding, and aggregate verification.

Second, KSDAuth adopts a 3-of-3 private-key sharding model. The signing capability originally associated with the IdP signing key is divided into three private-key shares held by the SP, Auth, and Issuer. A single share is insufficient to generate a valid credential.

Third, KSDAuth constructs a collaborative authorization process based on the role invocation chain of cross-domain access. The SP, Auth, and Issuer each generate a partial signature over the same canonicalized identity context. The SP then reconstructs and verifies the aggregated signature.

Fourth, KSDAuth binds user identity information and request state to the signing process. The signed identity context includes not only user attributes but also request_id, session_id, nonce, audience, and expiration time. This binding prevents token fragments from different requests or different users from being mixed successfully.

Through this design, KSDAuth reduces the risk that a single compromised signing component can independently forge credentials. The design also makes the security assumptions and deployment trade-offs explicit: it improves resistance to single-share leakage, but it requires secure setup, authenticated communication channels, and SP-side state validation.

Architecture of KSDAuth

The architecture of KSDAuth contains two major stages: a pre-configuration stage and a distributed authorization stage. In the pre-configuration stage, the original private signing key is split into multiple private-key shards and assigned to the SP, Auth, and Issuer, denoted as sk1, sk2, and sk3, respectively. Only when all required shards are logically combined can the behavior of the original signing capability be reproduced. Importantly, the distributed authorization process itself does not require reconstructing the original private key explicitly.

Based on the original cross-domain access flow, KSDAuth introduces four additional operations. In operation I, when the SP receives a user request and determines that the user is not yet logged in, it constructs the canonicalized identity context ctx and signs ctx with its own private-key share sk1 to produce token1, which remains local to the SP. In operation II, after credential submission and verification begin at the IdP side, the Auth verifies the user credentials, confirms the corresponding request state, and signs the same ctx with sk2 to produce token2. In operation III, token2 is transmitted to the SP through an authenticated channel rather than being returned to the user. In operation IV, the SP receives token1, token2, and the user-facing token fragment from the Issuer, denoted as token3, and performs aggregate signature recovery and verification. If verification fails, the system identifies an authentication anomaly and rejects the request. The overall architecture and the added operations of KSDAuth are illustrated in Figure 6.

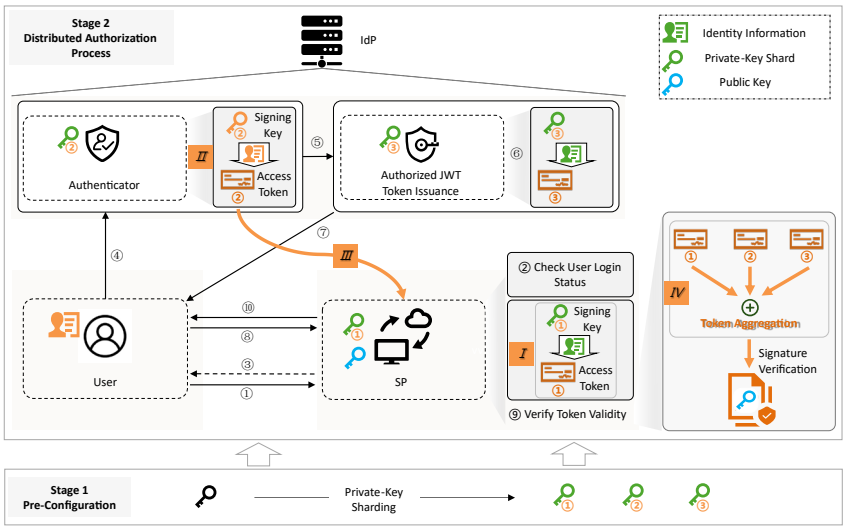

Figure 6. KSDAuth: Distributed Authorization Architecture Based on Private Key Sharding

Compatibility and Deployment Impact

KSDAuth should not be understood as a drop-in replacement for conventional JWT/SAML SSO systems. Its compatibility is mainly at the level of identity-claim semantics and credential formats. The SP can still process identity claims such as sub, iss, aud, exp, and privilege_set, but the signing and verification workflow is changed. In conventional SSO, the SP acts mainly as a verifier of credentials issued by the IdP. In KSDAuth, the SP also becomes a participant in the signing process because it holds one private-key share and generates a local partial signature. Therefore, existing SP implementations require an additional KSDAuth middleware layer to store the SP share, generate token1, receive token2 from Auth, receive token3 from the user-facing Issuer flow, and perform aggregate verification.

Table 1 summarizes the compatibility and deployment impact of KSDAuth.

Table 1 Compatibility and Deployment Impact of KSDAuth

Component	Conventional JWT/SAML SSO	KSDAuth	Deployment impact
Identity claims	Standard claims such as sub, iss, aud, exp	Preserved and included in ctx	Low
Signature generation	IdP/Issuer signs alone	SP, Auth, and Issuer generate partial signatures	High
SP role	Credential verifier	Partial signer and aggregate verifier	High

Component	Conventional JWT/SAML SSO	KSDAuth	Deployment impact
Auth role	User authentication	User authentication and partial signing	Medium
Issuer role	Token issuing	User-facing token-fragment issuing	Medium
Communication	Mainly browser redirection and token delivery	Additional Auth-to-SP token-fragment channel	Medium
Backward compatibility	Native	Requires middleware or protocol extension	Medium to high

Prototype Implementation

Secure Setup and Private-Key Sharding

To implement private-key sharding, this paper adopts a Lagrange-polynomial-based threshold model. The core idea of polynomial-based secret sharing is that k distinct points uniquely determine a polynomial of degree k−1. In a general t-of-n threshold setting, the secret is embedded as the constant term of a polynomial over the scalar field Fr of the BLS12-381 curve.

The general polynomial form is given in Equation (1):

$$f(x) = \mathrm{sk} + a_1 x + \cdots + a_{k-1} x^{k-1} \quad (1)$$

where sk is the master signing key, r is the group order of BLS12-381, and a1, a2, ..., a(t−1) are random coefficients sampled uniformly from Fr.

For the 3-of-3 case adopted in KSDAuth, t = 3. Therefore, a second-degree polynomial is used, as shown in Equation (2):

$$f(x) = sk + a_1 x + a_2 x^2 \bmod \mathrm{r} \quad (2)$$

The three private-key shares are generated by evaluating the polynomial at three distinct participant

identifiers:

$$sk1 = f(x1),\ sk2 = f(x2),\ sk3 = f(x3),\ \text{where } x1 = 1,\ x2 = 2,\ x3 = 3. \quad (3)$$

The shares sk1, sk2, and sk3 are assigned to the Service Provider (SP), the Authenticator (Auth), and the Token Issuer (Issuer), respectively. In this paper, KSDAuth adopts an n = 3, t = 3 threshold setting. This means that all three participants are required to contribute valid partial signatures before the aggregate signature can be reconstructed and verified. This design improves resistance to single-share leakage, but it also introduces an availability trade-off: if one participant is offline or if one share is lost, the authorization process cannot be completed.

Figure 7 illustrates the private-key sharding process using toy numerical values. These values are used only to explain the polynomial construction and must not be interpreted as real cryptographic parameters. In particular, the expression SecretKey(6) in the illustrative example does not represent a practical cryptographic key-generation method. In a real deployment, sk, a1, and a2 must be sampled uniformly from Fr. The current prototype assumes a trusted setup component that generates the secret shares in a protected environment, such as a hardware security module (HSM) or a trusted execution environment (TEE). Each share is delivered to the corresponding participant through an authenticated and encrypted channel. After distribution, the master signing key and the polynomial coefficients must be securely erased from the setup environment. This trusted-dealer model is a deployment assumption of the current design. A dealerless distributed key generation protocol could further reduce this assumption and is left as future work.

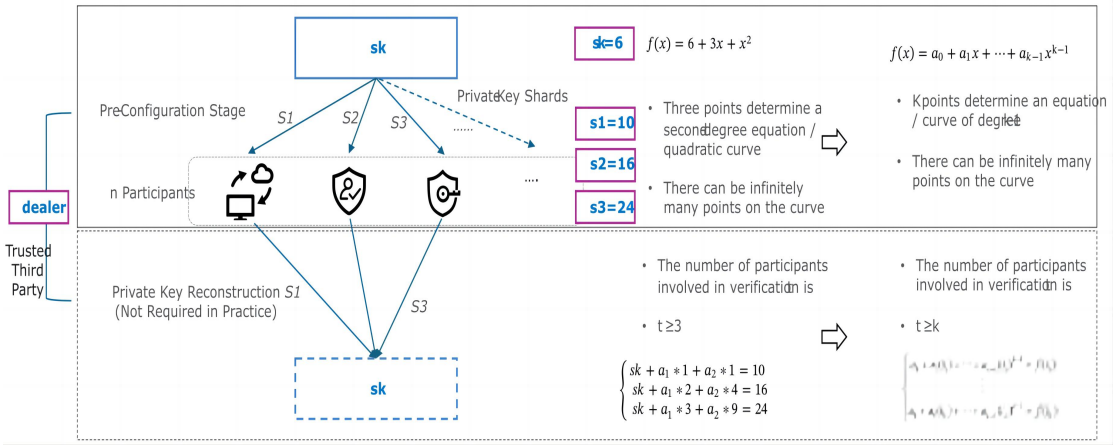

Figure 7. Example of Private Key Sharding

Algorithm 1 presents the private-key sharding process.

Algorithm 1. Practical Private-Key Sharding for KSDAuth

Input:

Security parameter λ

Scalar field Fr of BLS12-381

Participant identifiers x1 = 1, x2 = 2, x3 = 3

Output:

Public key pk

Private-key shares sk1, sk2, sk3

1. Initialize BLS12-381 parameters.

2. Sample the master signing key:

sk ←$ Fr

3. Compute the public key:

pk = g^sk

4. Sample random polynomial coefficients:

a1, a2 ←$ Fr

5. Define the degree-2 polynomial:

f(x) = sk + a1x + a2x^2 mod r

6. Generate private-key shares:

sk1 = f(x1)

sk2 = f(x2)

sk3 = f(x3)

7. Assign shares:

sk1 → SP

sk2 → Auth

sk3 → Issuer

8. Deliver each share through an authenticated and encrypted channel.

9. Securely erase sk, a1, and a2 from the setup environment.

Distributed Token Signing and Aggregate Verification

In KSDAuth, all participants sign the same canonicalized identity context ctx rather than loosely defined user information. The context ctx is derived from user identity attributes and request-state attributes as defined in Section 3.3. If any participant signs a different context, aggregate verification fails.

Algorithm 2 describes distributed token signing. Algorithm 3 shows the aggregate verification procedure.

Algorithm 2. Distributed Partial Signing

Input:

Identity context ctx

Private-key shares sk1, sk2, sk3

Output:

Partial signatures σ1, σ2, σ3

1. SP computes:

σ1 = Sign(sk1, ctx)

2. Auth computes:

σ2 = Sign(sk2, ctx)

3. Issuer computes:

σ3 = Sign(sk3, ctx)

4. SP keeps σ1 locally.

5. Auth sends σ2 to SP through an authenticated channel.

6. Issuer provides σ3 through the user-facing credential flow.

Algorithm 3. Aggregate Signature Recovery and Verification

Input:

Partial signatures σ1, σ2, σ3

Participant identifiers x1, x2, x3

Public key pk

Identity context ctx

Output:

verification_result $\in$ {True, False}

1. Check the freshness and source authenticity of σ1, σ2, and σ3.

2. Check that all partial signatures are bound to the same request_id and nonce.

3. Compute Lagrange coefficients λ1, λ2, λ3 for x1, x2, x3.

4. Recover the aggregate signature:

   σ = Π σi^λi

5. Verify:

   verification_result = Verify(pk, ctx, σ)

6. If verification_result is True, accept the credential.

7. Otherwise, reject the request and raise an authentication anomaly.

At the aggregate-verification stage, the SP receives the three token fragments and reconstructs a combined signature using the corresponding participant identifiers. The identifier list helps distinguish the contributions of the SP, Auth, and Issuer, and also provides traceability when verification fails. The reconstructed signature is then verified with the original public key. If the result is True, the collaborative authorization process is considered valid and the user is authenticated successfully. If the result is False, an authentication anomaly is detected, and the SP raises an alert and rejects the request.

SECURITY ANALYSIS AND EVALUATION

Defense Effectiveness Against Credential Forgery Attacks

Credential forgery based on stolen signing material is a practical security problem rather than a purely theoretical concern. Table 2 summarizes representative real-world incidents and attack tools involving forged identity tokens or credential artifacts. These cases are used only to motivate the threat addressed by KSDAuth. They should not be interpreted as experimental evidence that KSDAuth would have prevented each incident.

Table 2. APT Instances Related to Credential Forgery Attacks Based on Private Key Theft

APT Group / Tool	Type of Signing Key	Token Type	Attack Description
TA28	KRBTGT	Golden ticket	The attacker forged a TGT to access ICS services in an industrial enterprise network.
SolarWinds	ADFS signing certificate	SAML assertion	The attacker stole an ADFS signing certificate and forged a SAML assertion, compromising federated identity across organizations, including ICS-related environments.

APT Group / Tool	Type of Signing Key	Token Type	Attack Description
Storm-0588	MSA key	AAD token	The attacker obtained a Microsoft account signing key and forged an Azure AD token.
Dark Halo	Duo akey	Duo-sid cookie	The attacker used the Duo akey to forge a Duo-sid cookie and bypass MFA.
BeVigil	API key	OAuth token	The attacker used a stolen Twitter API key to take over Twitter accounts.
AAD Internals	AAD token-related signing material	AAD token	The attacker exposed ADFS signing and encryption certificates to a local directory environment.
Amazon AWS	Secret key	AWS STS token	The attacker used AWS credentials to request temporary security credentials through AWS STS.
Mimikatz	KRBTGT	Golden ticket	The attacker used the KRBTGT account secret to forge Kerberos TGTs and access endpoints in an ICS network.

Security Argument Under Single-Share Leakage

The main security claim of KSDAuth is single-share unforgeability under the defined threat model. Informally, an adversary who obtains only one private-key share cannot generate a valid aggregate signature over a new identity context unless the adversary can either forge missing partial signatures or compromise the remaining participants. Let A be a probabilistic polynomial-time adversary. A is allowed to obtain at most one private-key share ski and to request signatures on identity contexts observed during normal protocol execution. A wins the security game if it outputs a new identity context ctx* and a valid aggregate signature σ* such that Verify(pk, ctx*, σ*) = True, while ctx* was not jointly signed by SP, Auth, and Issuer. Under the existential unforgeability of the underlying BLS signature scheme and the assumption that fewer than three valid private-key shares are available to A, the probability that A wins this game is negligible in the security parameter λ. This is because producing σ* requires valid partial-signature contributions corresponding to the 3-of-3 participant set. A single compromised share is insufficient to reconstruct the aggregate signature for a new ctx*.

Analysis Under Single-Share Leakage

In the single-share leakage scenario, the adversary obtains one private-key share from SP, Auth, or Issuer, but does not control the other participants. The adversary may attempt to sign a forged identity context ctxA

using the compromised share. However, aggregate verification requires valid partial signatures from all three participants over the same ctxA. If the adversary compromises only the Issuer share, the adversary may generate σ3 = Sign(sk3, ctxA), but cannot generate σ1 and σ2. If the adversary reuses σ1 or σ2 from a legitimate request, those partial signatures are bound to a different request_id, nonce, session_id, or user identity. Therefore, the reconstructed signature will fail verification. The same reasoning applies when the compromised share belongs to SP or Auth. A single private-key share is not sufficient to produce a complete credential. This claim depends on two assumptions: first, token fragments are bound to the canonicalized identity context ctx; second, token-fragment transmission and freshness checking are correctly enforced by the SP.

Analysis Under Two-Share Compromise

The two-share compromise scenario represents a stronger adversarial capability. If the adversary obtains two private-key shares, the remaining honest participant becomes the last barrier against credential forgery. For example, if Auth and Issuer are compromised, the adversary can generate σ2 and σ3 over a forged identity context ctxA. The attack succeeds only if the SP also generates σ1 over the same forged ctxA. Therefore, the security of this case depends on SP-side state validation rather than on the threshold signature mechanism alone. The SP-side validation is not a generic identity check already present in conventional SSO. It is a request-state binding check. Before generating σ1, the SP must verify that:

• request_id was originally generated by the SP;

• nonce has not been used before;

• session_id is bound to the current authentication flow;

• audience matches the current SP;

• callback_uri matches the registered callback endpoint;

• privilege_set is derived from a trusted authorization source rather than user input;

• exp is within the permitted validity window;

• the Auth token fragment is received through an authenticated channel;

• the request has not been completed or replayed.

If these checks fail, the SP refuses to generate σ1 and aggregate verification cannot succeed. Therefore, KSDAuth provides conditional protection under two-share compromise only when the remaining honest

participant enforces strict request-state validation. If two compromised participants and the remaining participant all sign the same forged ctx, KSDAuth cannot prevent forgery.

Analysis Under Master Signing-Key Leakage

If the original master signing key is leaked, conventional JWT/SAML verification becomes insecure because the adversary can generate signatures that verify under the corresponding public key. Therefore, KSDAuth does not claim that master-key leakage is harmless.

KSDAuth mitigates this risk only when the SP enforces the distributed verification workflow and rejects credentials that do not contain valid evidence of the required partial-signature process. In other words, protected resources must disable legacy single-signature acceptance paths. If a legacy verification path remains enabled, an attacker with the master signing key may bypass KSDAuth by submitting a conventionally signed credential.

Therefore, master-key leakage should be treated as a critical incident. The appropriate response is key revocation, share regeneration, public-key update, and audit of all recently issued credentials.

Attack-Scenario Validation

Table 3 summarizes the expected behavior of KSDAuth under representative attack scenarios.

Table 3. Attack-Scenario Validation

Attack scenario	Compromised material	Expected KSDAuth behavior	Security condition
Issuer share leakage	sk3	Rejected	Missing σ1 and σ2 over the same ctx
Auth share leakage	sk2	Rejected	Missing σ1 and σ3 over the same ctx
SP share leakage	sk1	Rejected	Missing σ2 and σ3 over the same ctx
Payload tampering	Modified ctx	Rejected	Partial signatures bind different contexts
Replay of old token fragment	Old σi	Rejected	nonce and request_id freshness check
Auth + Issuer compromise	sk2, sk3	Conditionally rejected	SP refuses to sign forged ctx
SP + Issuer compromise	sk1, sk3	High risk	Requires honest Auth validation
All three shares compromised	sk1, sk2, sk3	Not protected	Out of scope
Master signing key leaked	sk	Critical	Legacy single-signature path must be disabled

Prototype-Level Performance Metrics

To guide prototype-level evaluation, this paper defines the following performance metrics for measuring the computational overhead introduced by KSDAuth: partial signing time, signature recovery time, aggregate verification time, end-to-end authorization latency, token-fragment size, and throughput. These metrics can be compared with the conventional single-signature JWT verification workflow in future empirical evaluation. Table 4 summarizes the metrics defined for prototype-level evaluation.

Table 4. Performance Metrics for Prototype Evaluation

Metric	Description
Partial signing time	Time for SP, Auth, or Issuer to generate one partial signature
Signature recovery time	Time for SP to reconstruct the aggregate signature
Verification time	Time for SP to verify the reconstructed signature
End-to-end latency	Total additional latency introduced by KSDAuth
Token-fragment size	Size overhead caused by transmitting partial signatures
Throughput	Number of authorization requests processed per second

The present manuscript focuses on mechanism design and security-boundary analysis. Large-scale performance evaluation under production workloads remains future work.

CONCLUSION

This paper presented KSDAuth, a distributed authorization mechanism based on private-key sharding for mitigating credential forgery caused by IdP signing-key leakage. KSDAuth replaces the conventional single-party signing model with a 3-of-3 collaborative signing model involving the SP, Auth, and Issuer. By requiring all participants to sign the same canonicalized identity context, KSDAuth reduces the risk that a single compromised signing component can independently forge a valid credential.

The revised threat model clarifies the adversarial capabilities, communication assumptions, node-compromise model, and security boundaries of the proposed mechanism. The security analysis indicates that KSDAuth can provide protection against single-share leakage under the assumptions of secure setup, authenticated communication channels, freshness checking, and SP-side request-state validation. The analysis also

shows that two-share compromise and master signing-key leakage require additional operational controls, such as strict SP validation, legacy-path disabling, key revocation, and share regeneration.

KSDAuth is not a zero-modification replacement for existing JWT/SAML SSO systems. Although it preserves identity-claim semantics, it requires additional protocol extensions or middleware for partial signing, token-fragment transmission, identity-context binding, and aggregate verification. The current work focuses on mechanism design, prototype-level evaluation metrics, and security-boundary analysis. Future work will include dealerless distributed key generation, broader empirical evaluation, production-scale deployment, interoperability testing, and optimization of the distributed authorization workflow.

Author Contributions

Conceptualization – H.M. and L.F.; methodology – H.M., L.F., and Q.Z.; formal analysis – H.M., L.F., and H.Y.; investigation – H.M., L.F., L.B., and H.Y.; resources – G.D.P.G. Digital Intelligent Operation Center team; writing—original draft preparation – H.M. and L.F.; writing—review and editing – H.M., L.F., and H.Y.; supervision – Q.Z. and Z.C. All authors have read and agreed to the published version of the manuscript.

Conflicts of Interest

The authors declare no conflict of interest.

Funding

This research was funded by the Major Science and Technology Special Project of China Southern Power Grid Headquarters, “Research and Application of Key Technologies for Cybersecurity Attack and Defense and Simulation in Digital Grids,” Project 4: “Research and Application Implementation Scheme of Multi-Dimensional Active Defense Technology for Digital Grids” (037800KC24040002, GDKJXM20240428).

Hazards

No unusual hazards were involved in this research.

Human Research Subjects

This research did not involve human subjects.

Animal Research Subjects

This research did not involve animal subjects.

Acknowledgements

The authors acknowledge the support provided by the Major Science and Technology Special Project of China Southern Power Grid Headquarters and the related research team of Guangdong Power Grid Co., Ltd.

REFERENCES

[1] Sandhu RS, Samarati P. Access control: principle and practice. IEEE Communications Magazine. 1994; 32(9):40-48. doi: 10.1109/35.312842

[2] Kubilay MY, Kiraz MS, Mantar HA. CertLedger: a new PKI model with certificate transparency based on blockchain. Computers & Security. 2019; 85:333-352. doi: 10.1016/j.cose.2019.05.013

[3] Alshamrani A, Myneni S, Chowdhary A, Huang D. A survey on advanced persistent threats: techniques, solutions, challenges, and research opportunities. IEEE Communications Surveys & Tutorials. 2019; 21(2):1851-1877. doi: 10.1109/COMST.2019.2891891

[4] Somorovsky J, Mayer A, Schwenk J, Kampmann M, Jensen M. On breaking SAML: be whoever you want to be. In: Proceedings of the 21st USENIX Security Symposium; 8-10 Aug 2012; Bellevue, WA, USA. Berkeley, CA: USENIX Association; 2012. p. 397-412. Available from: https://www.usenix.org/conference/usenixsecurity12/technical-sessions/presentation/somorovsky

[5] Armando A, Carbone R, Compagna L, Cuellar J, Tobarra M. Formal analysis of SAML 2.0 web browser single sign-on: breaking the SAML-based single sign-on for Google Apps. In: Proceedings of the 2008 ACM Workshop on Formal Methods in Security Engineering; 2008; Alexandria, VA, USA. New York, NY: Association for Computing Machinery; 2008. p. 1-10. doi: 10.1145/1456396.1456397

[6] Nugraha AF, Kabetta H, Buana IKS, et al. Performance and security comparison of JSON Web Tokens (JWT) and Platform Agnostic Security Tokens (PASETO) on RESTful APIs. In: 2023 IEEE International Conference

on Cryptography, Informatics, and Cybersecurity (ICoCICs); 22-24 Aug 2023; Bogor, Indonesia. Piscataway, NJ: IEEE; 2023. p. 15-22. doi: 10.1109/ICoCICs58778.2023.10277377

[7] Xiang C, Wu Y, Shen B, et al. Towards continuous access control validation and forensics. In: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security; 11-15 Nov 2019; London, United Kingdom. New York, NY: Association for Computing Machinery; 2019. p. 113-129. doi: 10.1145/3319535.3363191

[8] Zineddine A, Belfaik Y, Rehaimi A, Sadqi Y, Safi S. Single Sign-On Security and Privacy: A Systematic Literature Review. Computers, Materials & Continua. 2025; 84(3):4019-4054. doi: 10.32604/cmc.2025.066139

[9] Karie NM, Kebande VR, Ikuesan RA, Sookhak M, Venter HS. Hardening SAML by integrating SSO and multi-factor authentication (MFA) in the cloud. In: Proceedings of the 3rd International Conference on Networking, Information Systems & Security; 31 Mar-2 Apr 2020; Marrakech, Morocco. New York, NY: Association for Computing Machinery; 2020. p. 1-6. doi: 10.1145/3386723.3387875

[10] Ometov A, Bezzateev S, Mäkitalo N, Andreev S, Mikkonen T, Koucheryavy Y. Multi-factor authentication: a survey. Cryptography. 2018; 2(1):1. doi: 10.3390/cryptography2010001

[11] Li J, Fawaz K, Kim Y. Velody: nonlinear vibration challenge-response for resilient user authentication. In: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security; 11-15 Nov 2019; London, United Kingdom. New York, NY: Association for Computing Machinery; 2019. p. 1201-1213. doi: 10.1145/3319535.3354242

[12] Shafagh H, Burkhalter L, Ratnasamy S, Hithnawi A. Droplet: decentralized authorization and access control for encrypted data streams. In: 29th USENIX Security Symposium; 12-14 Aug 2020. Berkeley, CA: USENIX Association; 2020. p. 2469-2486. Available from: https://www.usenix.org/conference/usenixsecurity20/presentation/shafagh

[13] Andersen MP, Kumar S, AbdelBaky M, Fierro G, Kolb J, Kim HS, Culler DE, Popa RA. WAVE: a decentralized authorization framework with transitive delegation. In: 28th USENIX Security Symposium; 14-16 Aug 2019; Santa Clara, CA, USA. Berkeley, CA: USENIX Association; 2019. p. 1375-1392. Available from: https://www.usenix.org/conference/usenixsecurity19/presentation/andersen

[14]Philippaerts P, Preuveneers D, Joosen W. OAuch: exploring security compliance in the OAuth 2.0 ecosystem. In: Proceedings of the 25th International Symposium on Research in Attacks, Intrusions and Defenses; 26-28 Oct 2022; Limassol, Cyprus. New York, NY: Association for Computing Machinery; 2022. p. 460-481. doi: 10.1145/3545948.3545955

[15]Shoup V. Practical threshold signatures. In: Preneel B, editor. Advances in Cryptology – EUROCRYPT 2000. International Conference on the Theory and Application of Cryptographic Techniques; 14-18 May 2000; Bruges, Belgium. Berlin, Heidelberg: Springer; 2000. p. 207-220. doi: 10.1007/3-540-45539-6_15