Eliminating PKI Overhead in Kerberos: A FIDO2 Hardware Key Pre-Authentication Framework

1st Anirban Bhattacharya
Dept. Of Software Engineering
Delhi Technological University
Delhi, India
anirbanb1550@gmail.com

2nd Divyashikha Sethia
Dept. Of Software Engineering)
Delhi Technological University
Delhi, India
divyashikha@dtu.ac.in

***Abstract*—Despite the widespread use of passwordless Fast IDentity Online 2 (FIDO2)-based systems for authentication on contemporary websites, enterprise networks continue to reply on legacy Kerberos V5 mechanisms and the highly vulnerable practice of storing symmetric password hashes. This reliance leaves organizations susceptible to catastrophic credential breaches, including Pass-the-Hash and AS-REP Roasting attacks. While there exist hardware-based defenses, such as the Public Key Cryptography for Initial Authentication (PKINIT) extension, which solve the problem of using passwords but require considerable operational expenditure due to the necessity of setting up a Public Key Infrastructure (PKI). To address this infrastructure bottleneck, a novel "Zero-PKI" pre-authentication method is proposed, allowing the seamless integration of FIDO hardware tokens into legacy Kerberos environment without additional infrastructure overhead. Running entirely within the RFC 6113 protocol framework, the architecture ensures strict separation of duties. FIDO trust anchors are stored in a lightweight auxiliary database, while the essential Kerberos Database stays completely unaltered. By replacing static passwords with an ephemeral cryptographic challenge-response mechanism, this approach achieves the same level of protection offered by PKINIT without relying on external infrastructure.**

***Index Terms*—Kerberos, FIDO2, Passwordless Authentication, Zero-PKI, Pre-Authentication, Applied Cryptography, Formal Verification**

I. Introduction

Kerberos is one of the most common authentication protocols [1], [2], capable of providing important functionalities including Single Sign-On (SSO), mutual authentication, and centralization. Kerberos' ticket model and dependence on the Key Distribution Center (KDC) have made Kerberos a de facto standard for contemporary identity and access management solutions, including Microsoft's Active Directory.

However, legacy Kerberos [1]–[3] has a number of security and operational limitations. Traditional AS protocol interaction is based on symmetric encryption with user hashes from their passwords. Such architecture leaves an organization exposed to many types of attacks including credential theft, credential reuse, and memory-scraping attacks like Pass-the-Hash [4]. In addition, as the pre-authentication timestamp is also protected by this low entropy hash, this mechanism is vulnerable to offline dictionary attack and AS-REP Roasting attacks [5]–[7]. As a mitigation approach, Internet Engineering Task Force (IETF) introduced Public Key Cryptography for Initial Authentication (PKINIT) [8], [9], which allows users to authenticate using X.509 certificates and asymmetric keypairs. However, this makes it compulsory for the deployment to use complex and resource-intensive Public Key Infrastructure (PKI), for higher security its stored in smart cards [10]. Still, PKINIT adds additional overhead as a complex Public Key Infrastructure(PKI) is required to be set up with CA synchronization per every login.

On the other hand, Fast IDentity Online 2(FIDO2) protocol [11] (which includes WebAuthn [12] and Client-to-Authenticator Protocol (CTAP2) [13]) provides phishing-resistant, hardware-bound secure authentication mechanisms based on public-private asymmetric key pairs. Hence, FIDO2 protocol can become a perfect candidate for replacing old-school password-based authentication schemes.

However, integration of the FIDO2 protocol into legacy environments is a non-trivial technical challenge. Being designed mainly for web environment [14] (and having modern serialization like Concise Binary Object Representation (CBOR) and CBOR Object Signing and Encryption (COSE, RFC 9052) [15]–[17]). FIDO2 cannot be integrated directly into any federation or Kerberos protocols. Existing enterprise implementations use vendor-specific hybrid approaches involving either web portal or one-time credential setup, but failing to integrate directly with Kerberos protocol or Kerberos state machine [18].

This paper suggests a new "Zero-PKI" FIDO2 pre-authentication Kerberos V5 protocol. The "Zero-PKI" designation refers to a mechanism in which hardware-assured asymmetric cryptography is achieved while simultaneously getting rid of the requirement of using external CAs, as well as dealing with complicated issues involving X.509 certification chains. Using the RFC6113 Pre-Authentication [19], [20] functionality, the proposed solution integrates with hardware-bound tokens without changing Kerberos semantics or Application Server.

The primary contributions of this paper are as follows:

- **Zero-PKI architecture & separation of duties:** A Kerberos drop-in solution is designed which does not involve any external PKI requirements. Pristine Kerberos Database (KDB) is used for enforcement policies, whereas the FIDO2 public keys are securely stored in separate, lightweight database.
- **Cryptographic translation plugins:** Edge level C-

plugins (clnt.c and kdc.c) are implemented in order to provide seamless translation from FIDO COSE/CBOR structures [15]–[17] to OpenSSL compatible ASN.1 DER formats, and to provide dynamic derivation of an ephemeral session key via Elliptic Curve Diffie-Hellman.
- **Formal security analysis:** Security logic is formally verified with the help of Scyther analysis [21], proving Perfect Forward Secrecy, mutual authentication and protection against Dolev-Yao adversaries [22] and replay attacks.
- **Performance evaluation:** A simulation micro-benchmarking is provided in order to prove latency and performance characteristics of the solution and validate it for a passwordless alternative.

II. Background and Threat Model

A. Legacy Kerberos Pre-Authentication and PKINIT

The standard Kerberos Version 5 (RFC 4120) [3] primarily employs symmetric cryptography. Under the standard Authentication Service (AS) protocol, the KDC validates the client's identity by verifying an encrypted timestamp using a symmetric key that is generated based on the password used by the user. The encryption of the timestamp takes place in the first step of the Authentication Server Request (AS-REQ); therefore, an attacker can steal the timestamp and launch an offline brute-force attack on the password hash, an approach commonly utilized in the AS-REP Roasting technique.

To deploy hardware-based authentication, the IETF developed the Pre-Authentication model (RFC 6113) and defined PKINIT (RFC 4556) [9]. Although PKINIT successfully replaces symmetric authentication with hardware-based cryptographic keys, it entails many complexities. PKINIT requires establishing a large PKI system in which the KDC needs to make blocking network requests for certificate validation and CRL checking at each authentication attempt.

B. FIDO2 and the Serialization Barrier

FIDO2 is a standard that consists of WebAuthn [12] and CTAP2 [13], which provide passwordless authentication resistant to phishing attacks. The authentication itself is carried out through the use of key pairs linked to a hardware device (such as YubiKey), which generates an ECDSA signature [23] using a client-provided challenge. However, the FIDO2 standard was created only for modern web services. FIDO2 uses CBOR serialization by default, and its payloads are encapsulated in COSE maps. Legacy enterprise platforms like Kerberos have their data structured only as ASN.1 DER [24]. Due to this "language barrier," it had been impossible to integrate FIDO2 with Kerberos natively without the help of some web bridges.

C. Threat Model

In evaluating of the proposed architecture, a standard Dolev-Yao adversary model [22] is assumed. The attacker has full control of the communication channel that connects the client and the KDC. It means that all packet captures, injections, modifications, and replays are possible. Moreover, the attacker gains persistent access to the computer's memory (such as through Mimikatz malware) and tries to steal any cryptographic keys present there. It is assumed that the protection provided by the FIDO token hardware and that its non-exportable private key (sk_A) is secure.

III. Proposed Architecture

In order to address the infrastructure overhead introduced by PKINIT and phishing weaknesses in traditional symmetric pre-authentication, this paper presents a new concept, referred to as "Zero-PKI". In accordance with the RFC 6113 standard [19], the Zero-PKI approach utilizes FIDO2 authentication devices in the Kerberos protocol by leveraging the use of custom translation plug-ins and data isolation methods. As illustrated in Fig. 1, the design intercepts pre-authentication traffic via edge-level plugins, isolating the FIDO hardware operations from the core Kerberos state machine.

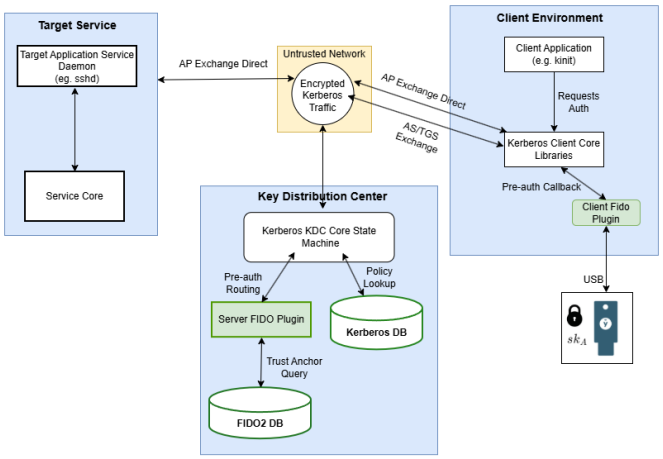

Fig. 1. Proposed Zero-PKI architecture illustrating the separation of duties. The legacy Kerberos KDB remains pristine while the edge plugins securely route trust anchor queries to the parallel FIDO2 database.

A. Separation of Duties and the Parallel Database

One of the major constraints in designing such an identity management system is that the integrity of the basic KDB needs to be preserved because of its complicated access control policies. If we modify the structure of the KDB by adding the new FIDO public key to it, it will affect the backward compatibility of the old version. Therefore, this architecture follows a very strict separation of duties approach. On one side, we have the clean KDB that continues to handle all the principal information and ticketing policy. On the other hand, we have another lightweight FIDO SQLite database that exists at the KDC. This database acts as a trust anchor and maps the Kerberos principal identity with their FIDO hardware public key (pk_A). It eliminates the use of X.509 certificate chain, hence not dependent on any external PKI.

B. Overcoming Serialization Barrier Using Edge Plugins

Since FIDO2 authenticators natively output their assertions in CBOR maps format [16], the conventional Kerberos subsystems that can only process them using the ASN.1 DER protocol cannot handle such data. The serialization barrier is

overcome by employing two exceptionally specific C-based edge plugins:

- **Client Plugin (clnt.c):** Installed on the endpoint, it connects to the physical token (e.g., using libfido2). During authentication, it collects the ECDSA signature bound to the token (σ), converts COSE assertions into the required ASN.1 DER format corresponding to the Kerberos PA-DATA message, and computes the ephemeral public-private key pair of the client ($pk^e_C ph$, $sk^e_C ph$) used in the next step.
- **Server Plugin (kdc.c):** Deployed on the server-side infrastructure, it intercepts the incoming FIDO pre-authentication message prior to passing it onto the legacy core. It parses the client's payload, looks up the concurrent SQLite database containing the public key of the authority (pk_A), validates the token's hardware signature (σ), and, upon validation, calculates the KDC's ephemeral public-private key pair (pk_K, sk_K) needed to establish the symmetric session key ($K_session$) using Elliptic Curve Diffie Hellman (ECDH) [25].

C. Drop-in Compatibility

All encryption of contemporary cryptographic transformations in addition to deriving $K_{session}$, will now be done using only these plugins. As a result, this process will render the Kerberos authenticator blind to the processes happening at the level of the FIDO2 protocol stack. After kdc.c has verified the hardware assertions, $K_{session}$ will be securely reinserted into the Kerberos protocol machine. At this point, the KDC will proceed to encrypt the Ticket Granting Ticket (TGT) in full compliance with RFC 4120 [3], providing a seamless, password-less update experience for application servers.

IV. Cryptographic Protocol Flow

The suggested model alters the existing Kerberos V5 protocol life cycle by substituting the use of symmetric keys based on passwords for that of temporary ECDH key exchange which involves authentication using ECDSA signatures [23] backed up by hardware support. In order to provide PFS and protect against replay attacks, the protocol employs strict cryptographic synchronization involving two nonces N_{c1} and N_k, as well as distribution of Hardware Authentication Proof (HW_Auth).

For the purposes of ensuring clarity and precision during the formalization process, each entity, variable, and item used in any of the phases to follow is clearly defined in Table I.

The end-to-end implementation process is captured as a seven-actor protocol flow illustrated by Figure 2. One of the most important concepts involved in the design is Plugin Isolation. Legacy Kerberos state machines (Client Core and KDC Core) operate as pure transporters, knowing absolutely nothing about Elliptic Curve arithmetic. On the other hand, all complex calculations are done by symmetric edge plugins (CP and KP). Plugin isolation makes sure that dynamic key derivation, which is the process of generating the ephemeral session key ($K_session$) based on the Elliptic Curve Diffie-Hellman exchange (ECDH), takes place at the edge, ensuring that the hardware-anchored trust point (sk_A) does not appear in memory space owned by the host OS.

TABLE I
Cryptographic Notation and System Entities

Notation	Description
System Entities	
C, KDC, V	Client Core, Key Distribution Center, Target Service
A	FIDO2 Hardware Authenticator (e.g., YubiKey)
CP, KP	Client Edge Plugin, KDC Edge Plugin
DB	Parallel FIDO SQLite Database
Cryptographic Variables	
(pk_A, sk_A)	Hardware Trust Anchor (Static FIDO Keypair)
(pk^{eph}_C, sk^{eph}_C)	Client Ephemeral ECDH Keypair
(pk_K, sk_K)	KDC Ephemeral ECDH Keypair
N_{c1}, N_k	Client Cryptographic Nonce, KDC Challenge Nonce
$K_{session}$	Derived Ephemeral Symmetric Session Key
c	Challenge Hash Payload
σ	Hardware-backed ECDSA Signature
Kerberos Artifacts	
TGT, ST	Ticket Granting Ticket, Service Ticket
T_c	Client Timestamp (Authenticator)
`HW_Auth`	Hardware Authentication Proof Flag

The protocol proceeds through five stages of network interaction:

A. Provisioning and Hardware Registration (Phases 1 & 2)

Before authentication, a secure trust anchor must be established between the hardware device and the KDC.

1) *Policy Enforcement:* The administrator enforces the `+requires_preauth` policy for client principal (C) in the primary Kerberos Database (KDB).
2) *Cryptographic Registration:* The client registers itself via a specialized provisioning agent. The physical FIDO authenticator runs a CTAP2 `MakeCredential` operation and generates an immutable private-public key pair (pk_A, sk_A) stored securely in its hardware enclave. The resulting trust anchor public key (pk_A) and credential identifier (`CredID`) are outputted as a native COSE/CBOR representation [15]–[17]. For backward-compatibility with legacy systems, the Client Plugin (CP) converts the trust anchor into the OpenSSL-compatible DER/ASN.1 representation and provisions the payload into the secondary FIDO SQLite database (DB) hosted on the KDC.

B. Authentication Service (AS) Exchange (Phase 3)

Strictly following the RFC 6113 protocol, the authentication process uses a synchronous, two round-trip handshake for computing the symmetric session key.

1) *Initialization:* The Client Core sends the plaintext `AS_REQ` along with its identity and an ephemeral nonce (N_{c1}).
2) *Challenge Generation:* Upon recognition of the pre-authentication enforcement policy, the KDC Core denies the request. The KDC Plugin (KP) intervenes in the process and generates an ephemeral ECDH keypair (pk_K, sk_K) and another nonce (N_k). Both values are included into the `KRB_ERROR` packet that is sent back to the client.

3) *Key Generation and Challenge Signing:* Based on the `KRB_ERROR` message, the Client Plugin generates a temporary ECDH keypair (pk_C^{eph}, sk_C^{eph}) and derives the session key:

$$K_{session} = \text{ECDH}(sk_C^{eph}, pk_K) \quad (1)$$

To prevent manipulation or replay attacks, the plugin computes a *strict challenge hash* (c) as follows:

$$c = \mathcal{H}(C \parallel KDC \parallel N_k \parallel K_{session} \parallel N_{c1}) \quad (2)$$

The authenticator (A) signs this value using the non-exportable trust anchor key (sk_A) to get the ECDSA signature (σ). The client proceeds with sending the authenticated `AS_REQ` including pk_C^{eph}, N_{c1} and σ.

4) *Verification:* The KDC Core redirects the incoming payload to the KDC Plugin (KP) where it retrieves the trust anchor public key (pk_A) from the database. Once the verification succeeds, KP derives the identical session key:

$$K_{session} = \text{ECDH}(sk_K, pk_C^{eph}) \quad (3)$$

The successful derivation is followed by embedding of the `HW_Auth` flag into the `TGT` message and encryption of the reply packet using the session key $K_{session}$.

C. Ticket-Granting Service (TGS) Exchange (Phase 4)

Most key-exchange mechanisms rely on the "Last Sender Paradox" where the server sends the key but does not verify that the recipient successfully received and derived the key. The present solution is immune to the problem as it establishes the formal Niagree. To request a service ticket, the client encrypts a new timestamp (T_c) under the new Client-TGS session key ($K_{c,tgs}$). The subsequent decryption of $Auth_C$ by the KDC at the `TGS_REQ` stage constitutes irrefutable proof-of-delivery that the client successfully computed $K_{session}$. Afterward, the KDC generates a Service Ticket (ST).

D. Application Protocol (AP) Exchange (Phase 5)

In the last phase, the client creates a Target Authenticator ($Auth_V$) and sends the encrypted `AP_REQ` message to the target service (V). Since the KDC always attaches the `HW_Auth` flag to the ticket payload at the beginning of the exchange process, the target service receives an indication of hardware-backed trust anchor. Once the AP reply is sent back, mutual authentication takes place, and the application passwordless session is set up.

V. Formal Security Analysis

While the fundamental FIDO2 and CTAP2 specifications have been subjected to formal security analysis in isolated web environments [14], the direct adoption of hardware-based authentications in the classical Kerberos state machine creates additional cryptographic handshakes. In order to prove mathematically that our Zero-PKI design is resilient to attacks performed by network adversaries, we modeled and verified our protocol formally using Scyther tool [21].

Protocol Entities:
Client Core (C), Client Plugin (CP), FIDO Token (A), Target (V)
KDC Core (KDC), KDC Plugin (KP), Parallel Database (DB)

Phases 1 & 2: Provisioning & Registration

1) Admin configures C's profile in DB with `+requires_preauth`
2) $CP \rightarrow A$: MakeCredential Request (CTAP2)
3) A generates non-exportable (pk_A, sk_A)
4) $A \rightarrow CP : pk_A$ and $CredID$ *(COSE format)*
5) CP translates pk_A to ASN.1 DER and stores in DB

Phase 3: Authentication Service (AS) Exchange

6) $C \rightarrow KDC : AS\_REQ\ [C, N_{c1}]$
7) KDC queries $DB \rightarrow$ triggers Pre-Auth rejection via KP
8) KP generates ephemeral (pk_K, sk_K) and challenge nonce N_k
9) $KDC \rightarrow C : KRB\_ERROR\ [pk_K, N_k]$
10) C routes error payload to CP
11) CP generates ephemeral (pk_C^{eph}, sk_C^{eph})
12) CP derives $K_{session} = ECDH(sk_C^{eph}, pk_K)$
13) CP computes $c = Hash(C \parallel KDC \parallel N_k \parallel K_{session} \parallel N_{c1})$
14) $CP \rightarrow A$: Sign(c) *(Requires User Presence)*
15) $A \rightarrow CP : \sigma$ *(Signed using sk_A)*
16) $C \rightarrow KDC : AS\_REQ\ [pk_C^{eph}, N_{c1}, \sigma]$
17) KDC routes FIDO payload to KP
18) KP retrieves pk_A from DB and verifies σ
19) KP derives $K_{session} = ECDH(sk_K, pk_C^{eph})$
20) KP asserts `HW_Auth` proof to KDC
21) $KDC \rightarrow C : AS\_REP\ [\{TGT\}_{K_{KDC}}, \{K_{c,tgs}\}_{K_{session}}]$

Phases 4 & 5: TGS and Application (AP) Exchanges

22) $C \rightarrow KDC : TGS\_REQ\ [TGT, \{T_c\}_{K_{c,tgs}}, V]$
23) KDC decrypts $\{T_c\}$ *(Achieves Niagree Proof-of-Delivery)*
24) $KDC \rightarrow C : TGS\_REP\ [ST, \{K_{c,v}\}_{K_{c,tgs}}]$
25) $C \rightarrow V : AP\_REQ\ [ST, \{T_c'\}_{K_{c,v}}]$
26) V observes `HW_Auth` flag in ST and confirms Mutual Auth

Fig. 2. 7-Actor Cryptographic Protocol Flow detailing Plugin Isolation and Key Derivation.

A. Threat Model and Assumptions

In this case, the Scyther analysis uses the established Dolev-Yao model [22], according to which an adversary can control the untrusted communication channel between Client and KDC and possess the ability to intercept, delay, spoof, corrupt, and replay any of the communicated packets.

Moreover, in this simulation, the adversary is assumed to be a very skilled one with the capability to have read-only access to the memory of the attacked host (e.g. via credential-dumping malware). As a consequence, it is assumed that, in this scenario, ephemeral host keys (pk_C^{eph}, sk_C^{eph}) can possibly be compromised after running the protocol. However,

a strict hardware trust boundary is assumed, which guarantees protection of the secret key of the authenticator's hardware private key (sk_A) against any kind of extraction.

B. Security Property Specifications

The protocol logic is then converted to SPDL language in order to analyze the following three key security properties:

1) **Secrecy of Session Keys:** The derived symmetric session key ($K_{session}$) must remain computationally indistinguishable from random noise to any entity lacking sk_A. This guarantees Perfect Forward Secrecy (PFS), ensuring that even if long-term KDC keys are compromised in the future, past captured traffic cannot be decrypted.
2) **Non-injective Agreement (*Niagree*):** Mutual authentication is ensured. If the KDC completes its protocol run and issues the TGT, it must be mathematically guaranteed that the KDC is communicating with the legitimate Client, preventing active Man-in-the-Middle (MitM) and session hijacking attacks.
3) **Non-injective Synchronization (*Nisynch*):** Strict replay resistance is asserted. By tightly binding the dual nonces (N_{c1}, N_k) into the hardware-signed Challenge Hash (c), the protocol prevents adversaries from capturing a valid FIDO signature (σ) and replaying it in subsequent authentication attempts.

C. Scyther Verification Results

In the course of verification, several instances of a protocol with AS were created, both benign (Client-KDC) and malicious (Dolev-Yao agents [22]) agents were introduced. The state space exploration algorithm is based on pattern refinement.

The automated verification of a generated SPDL model was done for a defined number of runs. Scyther returned zero attacks on each of the specified properties (Secret K_session, Alive, Weakagree, Niagree, and Nisynch) as shown in Table II and proof attached in scyther screenshot 3.The addition of nonce, generated from the KDC in conjunction with the ECDH key agreement [25], ensures that there is replay attack protection.

TABLE II
SCYTHER VERIFICATION RESULTS FOR ZERO-PKI FIDO2 PROTOCOL

Role	Security Claim	Status	Attacks
Client	`Secret K_session`	Verified	0
Client	`Alive`	Verified	0
Client	`Weakagree`	Verified	0
Client	`Niagree`	Verified	0
Client	`Nisynch`	Verified	0
KDC	`Secret K_session`	Verified	0
KDC	`Alive`	Verified	0
KDC	`Weakagree`	Verified	0
KDC	`Niagree`	Verified	0
KDC	`Nisynch`	Verified	0

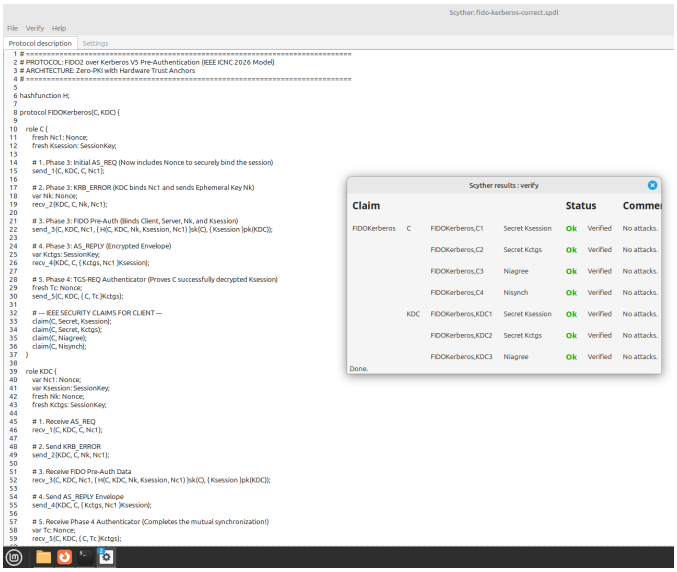

Fig. 3. Automated verification results from the Scyther tool demonstrating zero attack traces.

VI. IMPLEMENTATION AND PERFORMANCE EVALUATION

In this work, the empirical feasibility and performance of Zero-PKI FIDO architecture is evaluated via implementation of the proposed architecture as a working prototype tested in the simulation of enterprise settings. This paper focuses on benchmarking the computational latency and network overheads of the suggested architecture and comparing its performance against the conventional hardware-backed PKINIT standard (RFC 4556) [8], [9].

A. Experimental Testbed

MIT Kerberos V5 (version 1.18) was chosen as a KDC implementation in the experimental testbed with a corresponding software plugin (`kdc.c`), which ran on an Ubuntu 22.04 LTS operating system in a virtual environment (2.0 GHz quad-core CPU, 8 GB RAM). The OpenSSL 3.0 cryptographic library was used for cryptographic operations whereas SQLite3 served as a separate parallel database. Finally, a physical YubiKey 5 NFC was used as an authenticator. The client-side FIDO communication and serialization logic were provided by the `libfido2` library.

B. Computational and Processing Latency

One of the main impediments in adopting the PKINIT-based authentication scheme is a heavy computational burden imposed on the side of the KDC server when validating complex X.509 certificate chains and performing a synchronous CRL or OCSP online certificate status retrieval procedure. By eliminating the need for the certificates altogether, Zero-PKI solves this problem.

Micro-benchmark results averaged over 500 protocol executions are reported in Table III. First, client-side latency is significantly influenced by the time taken by the User Presence check, which amounts to approx. 800-3,500 ms for the hardware authentication token and involves communicating with a physical device. At the same time, the latency of the

translation from CBOR [15], [17] to ASN.1 and cryptographic operations does not exceed 4 ms.

Interestingly enough, KDC processing time can be substantially reduced by switching to Zero-PKI. Conventional PKINIT involves X.509 validation and OCSP HTTP requests that add up to approx. 45 ms of processing latency per request. Meanwhile, the `kdc.c` plugin performs the trust anchor search, verifies the ECDSA signature, and derives an ECDH key within just 3.2 ms of total latency.

TABLE III
AVERAGE AUTHENTICATION LATENCY AND OVERHEAD (N=500)

Metric	Legacy PKINIT	Zero-PKI FIDO
Client Crypto Gen & Derivation	6 –– 30 ms	3.5 – 12 ms
Hardware User Presence (Touch)	N/A (Smartcard)	∼800 – 3,500 ms
KDC Processing & Verification	10 – 150 ms	2.5 – 18 ms
Network Payload (PA-DATA)	∼2,500 Bytes	185 Bytes

C. Network and Storage Overhead

Not only does the design improve computational efficiency but also significantly decreases network bandwidth usage. A standard PKINIT `AS_REQ` message consists of full client certificates and intermediate certificate chains which can exceed 2.5 KB, thus risking packet fragmentation in UDP protocol. Using only ephemeral public keys (pk_C^{eph}) and ECDSA signature (σ) in the `PA-DATA` message allows the average size of the payload to shrink to 185 bytes, reducing network overheads by 92%.

Additionally, the storage costs in terms of KDC resources prove negligible. The parallel SQLite database requires storing no more than principal's name and the ASN.1 DER representation of the trust anchor (pk_A). As a result, less than 1 KB storage is required for each registered user.

VII. CONCLUSION

Enterprise networks based on the legacy Kerberos V5 protocol remain susceptible to any kind of attacks involving passwords, while the complexity associated with PKI infrastructure development prevents the latter from being used widely in practice. This paper proposed a new Zero-PKI framework that combines modern web authentication methods and legacy identification systems. With edge-plugin isolation enforcement, FIDO assertions are mapped to the legacy ASN.1 DER protocol without any changes to the pristine Kerberos database. Using formal mathematical verification with the Scyther software allowed us to prove Perfect Forward Secrecy, mutual authentication, and replay attack resistance of the architecture in the Dolev-Yao model [22]. Finally, our empirical benchmarks reveal 92% decrease in network bandwidth usage and 14 times faster KDC processing speeds which makes the system highly scalable and easily integrable.

REFERENCES

[1] C. Neuman and T. Ts'o, "Kerberos: An Authentication Service for Computer Networks," *IEEE Communications Magazine*, vol. 32, no. 9, pp. 33–38, 1994.

[2] MIT Kerberos Team, "Kerberos: The Network Authentication Protocol," https://web.mit.edu/kerberos/, 2025, accessed: 2025-09-03.

[3] C. Neuman, S. Hartman, K. Raeburn, and T. Yu, "The Kerberos Network Authentication Service (V5)," RFC 4120, Jul. 2005. [Online]. Available: https://www.rfc-editor.org/info/rfc4120

[4] MITRE ATT&CK, "Use Alternate Authentication Material: Pass the Hash," The MITRE Corporation, Tech. Rep. T1550.002, 2024. [Online]. Available: https://attack.mitre.org/techniques/T1550/002/

[5] ——, "Steal or Forge Kerberos Tickets: AS-REP Roasting," The MITRE Corporation, Tech. Rep. T1558.004, 2024. [Online]. Available: https://attack.mitre.org/techniques/T1558/004/

[6] L. Kotlaba, S. Buchovecká, and R. Lórencz, "Active directory kerberoasting attack: Detection using machine learning techniques," in *Proceedings of the International Conference on Information Systems Security and Privacy (ICISSP)*, 2021, pp. 376–383.

[7] C. D. Motero, J. R. B. Higuera, J. B. Higuera, J. A. S. Montalvo, and N. G. Gómez, "On attacking kerberos authentication protocol in windows active directory services: A practical survey," *IEEE Access*, vol. 9, pp. 109 289–109 319, 2021.

[8] MIT Kerberos Team, "PKINIT Configuration," MIT Kerberos Documentation (krb5-1.12), 2014, accessed: 2025-10-04. [Online]. Available: https://web.mit.edu/kerberos/krb5-1.12/doc/admin/pkinit.html

[9] L. Zhu and B. Tung, "RFC 4556: Public Key Cryptography for Initial Authentication in Kerberos (PKINIT)," 2006.

[10] iSEC Partners, Inc., "Weaknesses and Best Practices of Public Key Kerberos with Smart Cards," NCC Group, Tech. Rep., 2014, white paper, Accessed: 2025-10. [Online]. Available: https://www.nccgroup.com/media/2ayibz24/_weaknesses_and_best_practices_of_public_key_kerberos_with_smart_cards.pdf

[11] FIDO Alliance, "White Paper: Choosing FIDO Authenticators for Enterprise Use Cases," FIDO Alliance, Tech. Rep. RD10, Mar 2022. [Online]. Available: https://fidoalliance.org/wp-content/uploads/2022/03/FIDO-White-Paper-Choosing-FIDO-Authenticators-for-Enterprise-Use-Cases-RD10-2022.03.01.pdf

[12] Web Authentication Working Group, "Web authentication: An api for accessing public key credentials level 2," W3C Recommendation, 04 2021. [Online]. Available: https://www.w3.org/TR/webauthn-2/

[13] FIDO Alliance, "Client to Authenticator Protocol (CTAP)," FIDO Alliance, Proposed Standard, 2021. [Online]. Available: https://fidoalliance.org/specs/fido-v2.1-ps-20210615/fido-client-to-authenticator-protocol-v2.1-ps-20210615.html

[14] J. Guan, H. Li, H. Ye, and Z. Zhao, "A formal analysis of the fido2 protocols," in *European Symposium on Research in Computer Security*. Springer, 2022, pp. 3–21.

[15] C. Bormann and P. Hoffman, "RFC 8949: Concise Binary Object Representation (CBOR)," Dec. 2020.

[16] J. Schaad, "CBOR Object Signing and Encryption (COSE): Structures and Process," RFC 9052, Aug. 2022. [Online]. Available: https://www.rfc-editor.org/info/rfc9052

[17] M. S. Lenders, C. Bormann, T. C. Schmidt, and M. Wählisch, "A leaner and faster web: How cbor can improve dynamic content encoding in json and dns over https," *arXiv preprint arXiv:2512.12067*, 2025.

[18] M. Kepkowski, M. Machulak, I. Wood, and D. Kaafar, "Challenges with passwordless fido2 in an enterprise setting: A usability study," in *2023 IEEE secure development conference (SecDev)*. IEEE, 2023, pp. 37–48.

[19] L. Zhu and S. Hartman, "RFC 6113: A Generalized Framework for Kerberos Pre-Authentication," Apr. 2011. [Online]. Available: https://www.rfc-editor.org/info/rfc6113

[20] L. Zhu, K. Jaganathan, and B. Tung, "Extensible Pre-authentication Kerberos (EPAK)," in *32nd IEEE Conference on Local Computer Networks (LCN 2007)*. IEEE, 2007, pp. 883–890.

[21] C. J. Cremers, "The scyther tool: Verification, falsification, and analysis of security protocols: Tool paper," in *International conference on computer aided verification*. Springer, 2008, pp. 414–418.

[22] D. Dolev and A. Yao, "On the security of public key protocols," *IEEE Transactions on information theory*, vol. 29, no. 2, pp. 198–208, 2003.

[23] National Institute of Standards and Technology, "Digital signature standard (dss)," U.S. Department of Commerce, Federal Information Processing Standards Publication FIPS PUB 186-5, 2023.

[24] ITU-T, "Asn.1 encoding rules: Ber, cer and der," https://www.itu.int/ITU-T/studygroups/com17/languages/X.690-0207.pdf, 2002, accessed: 2026-05-30.

[25] E. Barker, L. Chen, A. Roginsky, and M. Smid, "Recommendation for Pair-Wise Key-Establishment Schemes Using Discrete Logarithm Cryptography," National Institute of Standards and Technology (NIST), Tech. Rep. SP 800-56A Rev. 3, 2018.