Security Research

|

Applied Cryptography

Private-Keyless, Passwordless, Time-Based Authentication

A Novel Protocol for Secrets-Free Identity Verification

"Eliminating shared secrets without sacrificing security."

Document Type	Technical Whitepaper
Version	1.0
Date	May 2026
Primary Author	Daniel Chien
Classification	Public Release — Research Draft

INTENDED AUDIENCE: PROTOCOL DESIGNERS · ENTERPRISE SECURITY ARCHITECTS · CISOS

Abstract

This paper presents a novel authentication protocol that eliminates both passwords and private cryptographic keys from the authentication flow entirely.

The scheme — designated *Private-Keyless, Passwordless, Time-Based Authentication* (PKTBA) — leverages precisely synchronized wall-clock time, obtained via hardened Network Time Protocol (NTP) infrastructure, as a coordinating ephemeral reference shared between client and server. Authentication tokens are derived deterministically from the current UTC time window, the user's identity, and an active method selected from a rotating server-defined array. Because both parties observe the same time window and apply the same public derivation function, no secret value ever needs to be exchanged, stored, or transmitted.

PKTBA is contrasted against TOTP (RFC 6238), FIDO2/WebAuthn passkeys, and magic-link systems. Unlike TOTP, PKTBA requires no per-user seed on either client or server. Unlike FIDO2, no private key material is ever stored on a device. The core security properties of the protocol include: an analog of forward secrecy (no persistent secret exists to steal), strict time-bounded token validity, stolen-device resistance, and cryptographic replay attack immunity via a per-user consumed-token ledger. The protocol's sole trust anchor is a hardened NTP layer secured using Network Time Security (NTS, RFC 8915), which is analyzed extensively. Threat modeling follows the STRIDE framework, and a reference implementation sketch is provided for protocol engineers.

Table of Contents

1. Introduction

1.1 The Credential Problem

Passwords and private cryptographic keys represent the primary attack surface of virtually every modern authentication system. Year after year, empirical breach data confirms that stolen or compromised credentials are the dominant initial access vector for malicious actors. The Verizon Data Breach Investigations Report (DBIR) 2024 found that compromised credentials were involved in over 80% of web application breaches, a figure that has remained stubbornly consistent across multiple reporting cycles. No amount of password policy enforcement, complexity requirements, or periodic rotation has meaningfully reduced this statistic at scale.

The response from the security industry has been to push toward "passwordless" authentication — most prominently embodied by FIDO2/WebAuthn and its consumer-facing form, passkeys. While these approaches eliminate the memorized password, they do so by substituting a *private asymmetric key* stored on the user's device. This private key, whether resident in a Trusted Platform Module (TPM), a Secure Enclave,

or a cloud-synchronized keychain, represents a new category of credential with its own attack surface: PIN extraction, device theft, cloud backup compromise, and platform-level vulnerabilities all become relevant threat vectors. The credential problem has been transformed, but not eliminated.

1.2 Motivation

The motivating goal of this work is to design an authentication protocol in which *no persistent secret exists on either the client or the server* that an attacker could steal and leverage for future authentication. A secret that does not exist cannot be breached, leaked, rotated, or phished.

The mechanism that makes this possible is time itself. Coordinated Universal Time (UTC), distributed globally via the Network Time Protocol (NTP), is universally accessible, continuously advancing, and — when properly secured — practically unforgeable in bulk. If both a client and a server independently observe the same UTC time window and apply the same deterministic derivation function to that time window and the user's public identity, they will arrive at the same ephemeral token without ever having communicated a secret. This insight is the foundation of the **Private-Keyless, Passwordless, Time-Based Authentication (PKTBA)** protocol described in this paper.

PKTBA shares a superficial resemblance to TOTP (RFC 6238) in its use of time windows, but diverges in a critical and fundamental respect: TOTP requires a per-user shared seed — a symmetric secret stored in both the authenticator application and the server's database. This seed *is* a secret, and its compromise is precisely the attack that has been used in large-scale OTP database breaches. PKTBA has no such seed.

1.3 Scope and Audience

This whitepaper is addressed to protocol designers, enterprise security architects, and Chief Information Security Officers (CISOs) evaluating next-generation authentication infrastructure. It covers the full design space of PKTBA: protocol mechanics, system architecture, NTP security hardening, threat modeling aligned to the STRIDE framework, brute-force resistance analysis, comparison with incumbent passwordless systems, and implementation guidance including reference

pseudocode. Formal security proofs are identified as future work; this document is intended as a comprehensive engineering and threat analysis suitable for security conference submission or enterprise evaluation.

2. Protocol Overview

2.1 Core Principle

PKTBA relies on precise, independently obtained time synchronization between the client and the authentication server. Both parties synchronize their clocks to UTC via hardened NTP infrastructure and, crucially, do so independently — there is no pre-authentication handshake that transmits a secret. Given a shared, publicly known derivation function and a shared, publicly known method array, both parties can independently compute the same ephemeral token for a given time window and user identity. The server validates the submitted token by reproducing this computation. No secret is ever exchanged.

The protocol's security properties derive entirely from the *ephemerality* of the computed token (it is valid for only one short time window), the *public inputs* to the derivation function (time and user ID — neither of which constitutes a secret), and the *replay prevention ledger* maintained server-side (which ensures each token can be used at most once). Because there is no secret input to the derivation function, there is no secret to steal. An observed token is useless after its window expires; a guessed token is defeated by token size, rate limiting, and the window's brevity.

2.2 Key Definitions

- **Time Window (T):** A fixed-duration epoch within which a token is valid. Computed as `T = floor(Unix_UTC / window_size)`, where the recommended `window_size` is 30 seconds, consistent with TOTP precedent (RFC 6238).
- **Method Array (M):** A server-defined, versioned, ordered set of authentication method identifiers (e.g., HMAC-SHA-256, BLAKE3, SHA3-256 keyed hash). The array is public metadata — its contents are not secret. The

active method index is `i = T mod len(M)`, computed independently by both parties.

- **Token (τ):** A short-lived authentication value computed as `τ = Derive(T, UserID, M[T mod len(M)])`. The derivation function is a cryptographic hash or keyed MAC, where any key material is itself derived ephemerally from `T` and `UserID` — never from a long-lived secret.
- **Ephemeral Key (K_{eph}):** For HMAC-based derivation methods, a per-window pseudokey computed as `Keph = H(T ‖ UserID)`. This key is unique to each token window and is not stored.
- **Clock Offset Tolerance (Δ):** A permitted clock drift range, expressed in number of time windows. The recommended default is $\Delta = \pm 1$ (the server accepts tokens for window $T-1$, T, and $T+1$), accommodating network latency and minor NTP drift.
- **Replay Prevention Ledger:** A server-side, per-user, per-window store of consumed token values. Entries expire after `window_size × (1 + 2Δ)` seconds (default: 90 seconds).

2.3 High-Level Authentication Flow

The following numbered sequence describes a complete authentication event under PKTBA:

1. Client and server each independently obtain current UTC time via their respective hardened NTP sources.
2. Client computes the current time window: `T = floor(Unix_UTC / 30)`.
3. Client determines the active derivation method from the public method array: `method = M[T mod len(M)]`.
4. Client computes the ephemeral token: `τ = Derive(T, UserID, method)`.
5. Client submits the tuple `(UserID, τ)` to the server over a TLS 1.3-protected channel.

6. Server independently computes `τ'` for windows `T−Δ` through `T+Δ` using the same derivation logic.
7. Server performs a constant-time comparison of `τ` against each candidate `τ'`.
8. On match: server checks the replay ledger. If the token is not yet consumed, it is marked consumed, and a session token is issued. If already consumed, authentication is rejected.
9. On no match across all tolerance windows: authentication fails. Rate limiting is applied to the failure.

Figure 1. *PKTBA Authentication Flow — Sequence Overview*

Step	Actor	Action	Notes
1	Client	Sync clock via NTP/NTS	Independently; no server contact at this stage
2	Client	Compute T = floor(UTC / 30)	30-second window index
3	Client	Select M[T mod len(M)]	Public method array; no secret
4	Client	Compute τ = Derive(T, UserID, method)	Ephemeral; no stored key
5	Client → Server	Send (UserID, τ) over TLS 1.3	No secret in transit
6	Server	Sync clock via NTP/NTS; compute τ' for T−1, T, T+1	Independent NTP sync
7	Server	Constant-time compare τ == τ'	Prevents timing side-channels
8	Server	Check replay ledger; mark consumed	Prevents replay within window
9	Server → Client	Issue session token (on success) or reject (on failure)	Rate limiting applied on failure

3. System Architecture

3.1 Component Model

PKTBA is composed of five logically distinct components:

- **Client Module:** Runs on the user's device (web browser, mobile application, or native client). Responsible for NTP clock synchronization verification, time window calculation, method array lookup, token derivation, and submission of the authentication request. The client module does not persist any secret material between sessions.
- **Authentication Server:** Receives authentication requests, maintains the method array and its version history, independently synchronizes its clock via NTS-hardened NTP, performs token validation, and manages the replay prevention ledger and session issuance. The server stores only user identities, the replay ledger (ephemeral), and optionally device binding metadata (public identifiers).
- **NTP Synchronization Layer:** Both client and server independently synchronize to a hardened NTP infrastructure. On the server side, this must be NTS (RFC 8915) using Chrony 4.x or NTPsec. On the client side, the OS NTP daemon is the primary mechanism, with an optional secondary server-provided timestamp sanity check.
- **Method Array Registry:** A server-maintained, versioned, and cryptographically signed list of permissible derivation method identifiers. This registry is public — clients download it at session start. The signing key is the sole long-lived cryptographic key in the system, stored in an HSM, and is used only for signing public metadata, not for authentication.
- **Replay Prevention Ledger:** A short-lived, per-user, per-window store that records consumed token values. Implemented as an in-memory cache (e.g., Redis with TTL) or a write-ahead log with automatic expiry. Entries survive for `window_size × (1 + 2Δ)` seconds and are then automatically purged.

3.2 Data Flow

Table 1. *PKTBA System Data Flow — Step-by-Step*

Step	Actor	Data In	Data Out	Notes
1	Client	NTP time response (authenticated)	Synchronized UTC timestamp	Via OS NTP daemon; NTS preferred
2	Client	UTC timestamp, WINDOW_SIZE=30	Time window T	T = floor(UTC / 30)
3	Client	Method array M (cached), T	Active method identifier	index = T mod len(M); array is public
4	Client	T, UserID, method	Token τ (64–128 bits)	Derive(); K_eph = H(T ‖ UserID)
5	Client → Server	(UserID, τ) over TLS 1.3	HTTP POST to auth endpoint	No secret transmitted
6	Server	NTP time response (NTS-authenticated)	Synchronized UTC timestamp	Independent of client NTP
7	Server	UserID, T±Δ, method array	Candidate tokens τ' for each window	Same derivation as client
8	Server	τ, τ' candidates	Boolean match result	Constant-time comparison
9	Server ↔ Ledger	UserID, T, τ	Consumed / Not consumed	Atomic check-and-set; TTL = 90s
10	Server → Client	Auth result	Session token (success) or error code	Audit log entry written

3.3 No Secrets at Rest

A defining architectural property of PKTBA is what is deliberately *absent* from persistent storage:

Not Stored — By Design

- No passwords (client or server)
- No private asymmetric keys (client or server)
- No shared symmetric secrets (neither per-user nor global)
- No per-user seeds or enrollment secrets (compare: TOTP requires a seed stored on both sides)
- No recovery codes or backup credentials of any kind

What *is* stored is strictly non-secret and minimal:

- **User IDs** — public identifiers, not secrets.
- **Replay ledger** — ephemeral, per-window consumed-token records. Automatically expire and contain no secret material.
- **Method array** — public, versioned, signed metadata.
- **Session tokens** — issued post-authentication, following standard session management practices (e.g., NIST SP 800-63B).
- **Device binding identifiers** (optional) — public device fingerprints or hardware attestation certificates; not secrets.

The method array signing key is the sole long-lived cryptographic key in the entire system. It is never involved in per-user authentication; its sole function is to authenticate the integrity of the public method array metadata.

4. NTP-Based Clock Synchronization

4.1 Role of Time in the Protocol

In PKTBA, synchronized UTC time is the coordinating factor that replaces the shared secret found in every other authentication system. If client and server agree on the current UTC time window to within Δ, they will independently derive the same token — without ever having communicated a secret value. This makes the reliability, accuracy, and security of clock synchronization directly load-bearing for the protocol's correctness and its resistance to attack.

Unlike shared secrets, time has a fundamental property that makes it uniquely suitable for this role: it cannot be stored by an attacker in the conventional sense. An attacker who observes the current time window gains nothing that will be useful in a future window, because time is continuously advancing. This temporal one-wayness is the source of PKTBA's forward-secrecy analog: past compromised tokens provide no leverage for future authentication.

4.2 NTP Fundamentals (RFC 5905, NTPv4)

The Network Time Protocol version 4 (NTPv4), specified in RFC 5905, distributes UTC time across hierarchically organized strata. Stratum 0 devices are primary reference clocks — typically GPS receivers, cesium atomic clocks, or WWVB radio receivers — that maintain time to within nanoseconds of UTC. Stratum 1 servers are directly connected to Stratum 0 devices and distribute time to Stratum 2 servers, which in turn serve clients.

NTPv4 achieves sub-millisecond synchronization accuracy in LAN environments and typically maintains accuracy within ±50 milliseconds on the public Internet under normal conditions. Client poll intervals range from 64 to 1024 seconds in steady state. The protocol uses a discipline algorithm that smooths clock adjustments using a phase-locked loop (PLL) model, preventing sudden large clock steps that could disrupt time-sensitive applications.

4.3 Clock Drift and Tolerance

The protocol must accommodate clock drift arising from NTP propagation delay, polling interval gaps, and network jitter. The following parameters are recommended:

- **Time window size:** 30 seconds (consistent with TOTP, RFC 6238).
- **Allowed drift tolerance (Δ):** ±1 window. The server evaluates tokens for windows T−1, T, and T+1, providing a total valid window of up to 90 seconds.
- **Maximum tolerated drift before re-sync trigger:** <15 seconds. If the protocol detects drift exceeding this threshold (via server-provided timestamp at login), it returns a `SYNC_REQUIRED` error code and defers authentication.
- **High-security deployments:** Δ may be reduced to ±0 (single 30-second window) when NTP reliability is guaranteed, minimizing the replay exposure window to 30 seconds.

4.4 Handling Offline Clients

Mobile devices and air-gapped systems maintain time between network connectivity windows using their hardware Real-Time Clock (RTC). Modern smartphone RTCs exhibit drift rates on the order of 0.5–2 parts per million (ppm), which translates to approximately 1.5–6 seconds of drift per month. For typical PKTBA deployments with Δ = ±1, brief offline periods of hours to days are well within tolerance.

For extended offline periods (weeks or more), the protocol mandates re-synchronization via NTP before authentication is attempted. The client module checks for clock drift on connection establishment: if local time deviates from the server-provided reference timestamp by more than Δ × window_size, the client delays authentication, triggers NTP re-synchronization, and retries. The server returns a structured `SYNC_REQUIRED` error with the server's current UTC timestamp (not a secret) to assist the client in calibration.

4.5 Time Window Boundary Attacks

A timing risk arises at window boundaries. A token computed one second before the end of time window T is valid during window T, but due to Δ = ±1 tolerance, it may also be accepted during window T+1 — extending its effective validity to up to 60 seconds beyond its computation time. This creates a narrow but non-trivial replay

window for an attacker who intercepts the token in transit at the precise moment of boundary transition.

Mitigations include:

- **Replay ledger (primary mitigation):** Each token is consumed on first use regardless of window boundary. A boundary-straddling token, once used, cannot be replayed in either window.
- **Reduced Δ in high-security deployments:** Setting $\Delta = \pm 0$ eliminates the boundary extension entirely at the cost of strict NTP reliability requirements.
- **Sub-window nonces:** An optional extension adds a sub-window nonce (e.g., a monotonic counter within the window) as an additional input to the derivation function, further reducing boundary collision risk.
- **TLS 1.3 transport:** Intercepting a token in transit requires breaking TLS 1.3, which is not feasible under current threat models.

5. Internal NTP Security

5.1 Why NTP Is a Critical Attack Surface

Because PKTBA's security derives from time agreement between client and server, the NTP synchronization layer is the protocol's most important trust anchor. An adversary who can manipulate either party's clock can mount two distinct categories of attack: (1) extend the valid window of a previously captured (expired) token by advancing the victim's clock backward; or (2) predict future valid tokens by advancing the server's clock forward, pre-computing expected tokens, and submitting them before the legitimate window arrives. Both attacks require the ability to inject false NTP responses — a well-documented and practically demonstrated capability.

5.2 Known NTP Attack Vectors

- **Clock Stepping / Time-of-Day Attacks:** An attacker manipulates NTP responses to advance or retard the victim's clock. On legacy NTP

implementations, large clock steps can be injected via a single forged NTP packet. This can open replay windows for expired tokens or invalidate legitimate tokens by causing apparent time reversal.

- **Man-in-the-Middle NTP Spoofing:** In the absence of NTP authentication, an on-path attacker can intercept and modify NTP packets in transit, substituting arbitrary timestamp values. This attack is particularly relevant on untrusted network segments (e.g., public Wi-Fi, compromised routers).
- **Kiss-o'-Death (KoD) Packet Abuse:** An attacker forges KoD control packets from the NTP server's IP address, causing the victim client to stop polling its NTP server entirely. This degrades time accuracy over time without the attacker needing to forge time values — eventual RTC drift opens replay windows.
- **NTP Amplification (Reflection DDoS):** Abuses NTP's monlist command to generate amplified reflection attacks. While not directly targeting authentication, it degrades NTP infrastructure availability, increasing clock drift across affected clients.
- **Rogue NTP Server Injection:** An attacker advertises a malicious NTP server via DHCP option 42 or a compromised router, directing clients to a clock source under attacker control.

5.3 Mitigations

Table 2. *NTP Attack Vectors and Protocol Mitigations*

Attack Vector	Mitigation	Standard / Reference
MITM NTP Spoofing	Network Time Security (NTS): authenticated, encrypted NTP over TLS 1.3 with AEAD. Eliminates unauthenticated packet injection entirely.	RFC 8915
Clock Stepping	Drift rate limiting: maximum clock	NTPsec, chrony slew-only mode

Attack Vector	Mitigation	Standard / Reference
	adjustment of 128ms per step; large adjustments require operator confirmation. Monotonic clock used for replay ledger TTL.	
Single-Server Compromise	Query 3–5 independent NTP sources; use Marzullo's intersection algorithm to reject outliers. A single compromised source cannot override the consensus.	RFC 5905 §11.2.1
KoD Packet Abuse	NTS eliminates unauthenticated KoD packets; NTPsec validates KoD authenticity before acting on it.	NTPsec project
Rogue DHCP NTP Injection	Hardcode NTP server addresses in client configuration; do not rely solely on DHCP-provided NTP servers. NTS certificate validation prevents impersonation even if server address is poisoned.	NTS PKI model
External NTP Dependency	High-security server deployments: on-premises GPS-disciplined Stratum 0/1 reference clock. Eliminates all reliance on external NTP infrastructure.	GPS PPS discipline

NTPsec (ntpsec.org) is the recommended NTP daemon for all PKTBA server deployments. NTPsec is a security-focused fork of the NTP reference implementation that eliminates legacy vulnerable code paths, implements memory-safe constructs, reduces attack surface by removing unused features, and supports NTS natively. Its codebase has undergone multiple independent security audits.

Network Time Security (NTS, RFC 8915) provides cryptographically authenticated, encrypted NTP using TLS 1.3 for the key exchange phase and AEAD (AES-128-GCM-SIV or ChaCha20-Poly1305) for packet authentication. NTS completely eliminates MITM and spoofing attacks against the NTP layer. It is supported by NTPsec ≥ 1.2.0, Chrony ≥ 4.0, and major public NTP providers including Cloudflare's time.cloudflare.com and the NIST NTS service.

5.4 Client-Side NTP Hardening

Client-side clock synchronization relies on the platform OS NTP daemon:

- **iOS / Android:** Platform-maintained synchronized clocks via Apple Time Service and Google Public NTP respectively. Both use authenticated NTP variants and are not configurable by the user (preventing accidental misconfiguration).
- **Windows:** W32Time service; upgrade to a third-party NTP client (e.g., Meinberg NTP) with NTS support for PKTBA deployments requiring high clock assurance.
- **Linux / macOS:** Chrony 4.x with NTS enabled is the recommended daemon.
- **Web clients (browser-based):** JavaScript's `Date.now()` draws from the OS NTP-synchronized clock. The PKTBA client module should additionally perform a server timestamp sanity check at login page load: if local time deviates from the server's `X-Server-Time` response header by more than $\Delta \times$ window_size, the module warns the user and delays authentication until the discrepancy is resolved.

6. Method Array Rotation

6.1 Purpose

The method array provides three distinct benefits to the PKTBA protocol: *algorithm agility* (the ability to rotate cryptographic primitives without protocol disruption),

derivation diversity (successive tokens use different derivation functions, complicating cross-window pattern analysis), and *defense-in-depth* (an attacker exploiting a weakness in one derivation method does not compromise tokens produced by other methods in the array). It is critical to understand that the method array is *not* a secrecy mechanism — its contents are public, and the protocol's security does not depend on keeping the active method hidden.

6.2 Method Array Design

The method array $M = [m_0, m_1, m_2, ..., m_{n-1}]$ contains distinct, named derivation method identifiers. Each identifier maps to a specific algorithm in the server's method registry. Example entries:

- `HMAC-SHA-256-v1` — HMAC with SHA-256; ephemeral key `K_eph = H(T ‖ UserID)`
- `HMAC-SHA-512-v1` — HMAC with SHA-512; same ephemeral key derivation
- `SHA3-256-KDF-v1` — SHA3-256 with HKDF-based key derivation
- `BLAKE2b-v1` — BLAKE2b-512 with domain separation prefix
- `BLAKE3-v1` — BLAKE3 keyed hash; ephemeral key as above

For HMAC-based methods, the "key" is not a long-lived secret. It is the ephemeral pseudokey `K_eph = H(T ‖ UserID)`, which is unique to each (window, user) pair and is never stored. The security of HMAC here derives from the unpredictability of the full input combination `(T, UserID, method)` to an attacker who does not control the user's identity or the clock, not from key secrecy.

6.3 Array Length and Rotation Frequency

The recommended method array length is 16 to 64 entries. With a 30-second window, a 16-method array completes one full rotation cycle every `16 × 30 = 480 seconds` (8 minutes). A 64-method array cycles every `64 × 30 = 1920 seconds` (32 minutes).

An eavesdropper observing all tokens emitted during one full rotation cycle observes every method variant in the array — but each token is already expired by the time the

cycle completes, providing no cryptographic leverage. The rotation frequency is calibrated to be useful for algorithm agility rather than for security through obscurity.

6.4 Array Versioning and Updates

The method array is versioned and signed by the server's identity key (stored in an HSM). When a new array version is published:

10. The server announces the new array version and its activation time (a future UTC timestamp) via the authenticated method array endpoint.
11. Clients download the new array and verify the server's signature before the switchover time.
12. During the transition period (recommended: 5 minutes), both the old and new arrays are accepted, preventing authentication lockouts.
13. After the transition period, the old array version is retired.

This mechanism supports cryptographic algorithm rotation (removing deprecated hash functions, adding post-quantum primitives) without disrupting active authentication sessions or requiring user re-enrollment.

6.5 Why Method Rotation Does Not Replace Secrets

Design Principle — Transparency Over Obscurity

The method array is public metadata. The protocol deliberately does *not* rely on hiding which derivation method is active. Security derives from the ephemerality of tokens (time-bounded validity), the temporal one-wayness of the time input, the size of the token space (128 bits), and the replay prevention ledger — not from method secrecy. This distinction is essential: security-through-obscurity assumptions would make the system fragile; transparency makes it auditable and robust.

7. Brute-Force Resistance

7.1 Token Space Analysis

The primary defense against brute-force attacks is token size. A 6-digit numeric token (TOTP-style) provides only 10^6 = 1,000,000 combinations — trivially exhaustible given a fast network connection and no rate limiting. PKTBA specifies a minimum token size of 64 bits and strongly recommends 128 bits, yielding a search space of $2^{128} \approx 3.4 \times 10^{38}$ combinations.

7.2 Time-Bounded Validity

PKTBA's time-bounded token validity provides an additional combinatorial constraint on brute-force attacks. With $\Delta = \pm 1$, a token is valid for at most 90 seconds. An attacker attempting to exhaust the token space must do so within this window before the token expires and an entirely new, independent token must be targeted.

Even assuming an extraordinarily fast attack rate of 10^9 guesses per second — far beyond what any authentication API would allow in practice — only $90 \times 10^9 = 9 \times 10^{10}$ values can be tested per window. Against a 64-bit token space (1.8×10^{19} values), this represents coverage of only 0.0000005% per window. Against a 128-bit space, the fraction is astronomically smaller.

7.3 Rate Limiting

The authentication server enforces layered rate limiting to make even the theoretical attack above impossible in practice:

- **Per-user rate limiting:** Maximum N failed attempts per time window (recommended: N = 5). After N failures, the account is locked for K windows (recommended: K = 10, i.e., 5 minutes).
- **Per-IP rate limiting:** Maximum M failed attempts per IP address per minute (recommended: M = 20). Applies independently of user-level limiting to prevent distributed credential stuffing.

- **Global anomaly detection:** Aggregate failure rate monitoring triggers circuit-breaker responses when global failure rates exceed baseline thresholds, indicating a coordinated attack campaign.
- **CAPTCHA gating:** After the first per-IP threshold breach, a CAPTCHA challenge is inserted to distinguish automated from human traffic.

7.4 Replay Prevention

The replay prevention ledger provides an additional defense dimension orthogonal to brute force. Even if an attacker correctly guesses a valid token, they must do so before the legitimate user submits it. Once submitted, the token is atomically marked consumed in the ledger and all subsequent submissions of the same value — whether by the legitimate user or an attacker — are rejected for the duration of the validity window. This atomic consume operation must be implemented using a distributed lock or compare-and-set (CAS) operation to prevent race conditions in multi-server deployments.

7.5 Token Size Recommendations

Table 3. *Token Size Analysis — Brute-Force Resistance at 10^9 guesses/second*

Token Size	Search Space	Time to Exhaust at 10^9 tries/sec	Recommendation
32 bits	4.3×10^9	~4.3 seconds	**Insufficient — Do Not Use**
48 bits	2.8×10^{14}	~3.2 days	**Marginal — Not Recommended**
64 bits	1.8×10^{19}	~586 years	**Minimum Recommended**
128 bits	3.4×10^{38}	Cosmological timescale	**Strongly Recommended**

All PKTBA implementations should default to 128-bit token output, truncated as a hex or base64-encoded string for transmission. Implementations requiring human-

readable tokens (e.g., voice-based authentication scenarios) should use a minimum 10-digit decimal code (10^{10} combinations) with enforced rate limiting.

8. Stolen-Device Protection

8.1 The Threat

Device theft or remote compromise is a fundamental threat to any authentication system. An adversary who controls a user's device may be able to observe the client application computing tokens in real time, extract any credentials stored persistently on the device, or use the device as an authentication oracle to generate valid tokens on demand. The severity of this threat depends critically on what secrets, if any, are resident on the device.

8.2 Why PKTBA Is Inherently Resistant

PKTBA's stolen-device resistance is structural rather than procedural. Since no long-lived secret is stored on the device — no seed, no private key, no shared symmetric key — an attacker who steals the device and extracts its entire storage gains nothing that enables authentication without the device being physically present and the current time window being active. The device can compute tokens, but only for time windows that are currently valid. There is no "offline attack" on stored material because there is no material to attack.

Furthermore, a stolen device that can compute tokens provides only 30 seconds of token validity per computation. The moment the device is reported stolen and the user's device binding is revoked server-side, all subsequent token submissions from that device are rejected — no key revocation ceremony, no re-enrollment of a new key, no waiting for certificate revocation propagation.

8.3 Device Identity Binding (Optional Layer)

For higher-assurance deployments, the protocol supports an optional device identity binding layer. A device-specific public identifier — such as a hardware attestation

certificate (Android SafetyNet/Play Integrity attestation, Apple DeviceCheck), a TPM Endorsement Key (EK) public certificate, or a device fingerprint hash — is registered with the authentication server at enrollment time. The server validates device identity as an additional factor during authentication.

Critically, the device identifier is a *public value*. It is not a secret. It provides a "something you have" factor without requiring the client to store a private key. If a device is stolen, the server simply removes the device binding entry, immediately and permanently invalidating all future authentication attempts from that device — even though no secret has been compromised.

8.4 Biometric Gate (Optional Layer)

A biometric authorization check (fingerprint reader, Face ID, iris scan) can be configured to gate the client module's ability to compute and submit tokens. This transforms PKTBA into a multi-factor protocol: something you have (the device, time-synchronized) plus something you are (biometric). Biometric data never leaves the device — it is evaluated by the platform's secure biometric processor (Secure Enclave on iOS, StrongBox on Android) and the result is a local boolean authorization, not a transmitted credential.

8.5 Session Invalidation

Upon reporting a device stolen, the server executes the following sequence:

14. Revokes all active sessions associated with the UserID across all devices.
15. Removes the stolen device's binding entry from the device registry (if device binding is in use).
16. Flags the UserID as requiring re-enrollment from a new device.
17. Logs the revocation event with timestamp and reporting channel for audit purposes.

No key revocation ceremony is required. There are no keys to revoke — only the ephemeral device binding entry, which is a public identifier.

8.6 Comparison to FIDO2 Stolen-Device Scenario

In FIDO2/passkeys, authentication requires the private key stored in the device's Secure Enclave. Stealing a device and obtaining the unlock PIN (via shoulder-surfing, rubber hose, or a malicious app that captures PIN entry) provides the attacker with the ability to perform unlimited authentications with the victim's identity — because the private key, once accessible, is a persistent capability. Revocation requires the user to recognize the compromise and manually revoke the passkey from each relying party.

In PKTBA, a stolen device with a known PIN only provides the ability to compute currently-valid time-window tokens — each valid for 30 seconds. As soon as the user reports the theft, server-side device binding revocation immediately terminates this capability. The attacker has a window measured in seconds to minutes before revocation, rather than an indefinite capability until manual key revocation is completed.

9. Threat Model

9.1 STRIDE Threat Analysis

Table 4. *STRIDE Threat Model — PKTBA Protocol*

STRIDE Category	Attack Scenario	Protocol Mitigation	Residual Risk
Spoofing	Impersonating a user without their device or real-time clock access	Token is bound to UserID and current time window. No static credential to steal. 128-bit token space defeats guessing.	Low. Attacker would need real-time system access.
Tampering	Modifying NTP packets to manipulate client or server clock	NTS (RFC 8915) provides authenticated, encrypted NTP.	Low with NTS. Medium without NTS.

STRIDE Category	Attack Scenario	Protocol Mitigation	Residual Risk
		Multi-source NTP + Marzullo's algorithm rejects outliers.	
Repudiation	User denies that an authentication event occurred	Server-side replay ledger records consumed token, time window, client IP, and session ID with UTC timestamps. Provides tamper-evident audit trail.	Low. Audit log provides non-repudiation evidence.
Information Disclosure	Intercepting a valid token in transit	Token single-use (replay ledger). Valid for ≤90 seconds. TLS 1.3 encrypts transport. Compromised token is worthless after expiry.	Very Low. Requires simultaneous MITM and sub-90s replay.
Denial of Service	Flooding the authentication endpoint	Per-user and per-IP rate limiting. CAPTCHA gating. Global anomaly detection. IP blocking.	Medium. DoS against rate limiting infrastructure is possible but impacts availability, not confidentiality.
Elevation of Privilege	Using a stolen valid token to escalate privileges beyond authenticated session	Token grants session only; privilege levels are checked post-authentication against the user's authorization profile. Token is session-bound.	Low. Depends on post-auth authorization model correctness.
NTP Clock Manipulation	Advancing server clock to re-open expired replay ledger entries	Replay ledger TTL based on server monotonic clock, not wall clock. NTS prevents clock injection. Drift rate limiting	Low with NTS + monotonic clock. Medium without.

STRIDE Category	Attack Scenario	Protocol Mitigation	Residual Risk
		bounds maximum clock step.	
Method Array Tampering	Serving a malicious method array to the client to cause predictable token derivation	Method array is signed by server identity key (HSM-protected). Client verifies signature before use. Invalid signature causes authentication abort.	Low. Requires compromise of server HSM or PKI.
Replay Attack	Re-submitting an intercepted valid token	Replay ledger atomically marks token as consumed on first successful use. Subsequent submissions of same token rejected for full validity window duration.	Very Low. Requires beating legitimate user to the server by milliseconds.
Brute Force	Exhaustive token guessing within a validity window	128-bit token space (3.4×10^{38} values). 90-second window. Per-user rate limiting (5 attempts/window). Account lockout after threshold.	Negligible. Mathematically infeasible within validity window.
Timing Side-Channel	Measuring server comparison timing to infer partial token values	Constant-time comparison required (`hmac.compare_digest` or equivalent). Prevents timing oracle attacks on token comparison.	Very Low with constant-time comparison. Critical if implementation omits this.

9.2 Trust Assumptions

PKTBA's security relies on the following assumptions. Protocol designers should explicitly evaluate each in their deployment context:

- The server's system clock is correctly maintained by NTS-hardened NTP with multiple independent sources.
- The client's OS NTP daemon is not compromised by the attacker (i.e., the client device's kernel is trusted).
- TLS 1.3 protects the authentication channel — specifically, certificate validation is correctly implemented and certificate pinning is enforced where applicable.
- The method array signing key is protected by a hardware security module (HSM) that has not been physically or logically compromised.
- The replay prevention ledger is implemented with atomic compare-and-set semantics and is not susceptible to race conditions in distributed deployments.

9.3 Out of Scope

The following threat categories are acknowledged but are outside the scope of this protocol specification:

- **Physical coercion:** An attacker compelling a user to authenticate under duress is a social engineering attack, not a protocol vulnerability.
- **Full server OS compromise:** An attacker with root access to the authentication server can manipulate the replay ledger, method array, and user store directly. This is equivalent to full infrastructure compromise and is outside the protocol's threat boundary.
- **Quantum attacks on SHA-256:** The second-preimage resistance of SHA-256 requires 2^{256} quantum operations under Grover's algorithm. Grover's algorithm squares the search cost, requiring 2^{128} classical-equivalent operations — still computationally infeasible. SHA-3 and BLAKE3 variants in the method array provide additional post-quantum margins.

10. Comparison to Traditional Passwordless Systems

Table 5. *Authentication System Comparison — Feature Matrix*

Feature	PKTBA (This Protocol)	TOTP (RFC 6238)	FIDO2 / WebAuthn / Passkeys	Magic Links	SAML / SSO
Private key stored on device	No	No	Yes (Secure Enclave / TPM)	No	No (IdP-side)
Shared secret stored on server	No	Yes (seed per user)	No	Temporary nonce	Yes (IdP credentials)
Seed / secret required for enrollment	No	Yes	Yes (key generation)	No	Yes
Token lifetime	30 seconds	30 seconds	Session-scoped (challenge-response)	Minutes to hours	Hours (configurable)
Replay protection	Yes — server-side ledger	Partial — server tracks last OTP	Yes — nonce-based challenge	Partial — if nonce tracked	Yes — assertion timestamps
Brute-force resistance	High (128-bit token space)	Low (6-digit: 10^6 space)	Very High (asymmetric crypto)	High (URL entropy)	High (IdP-enforced)
Stolen-device impact	Low — no persistent secret; instant server-side revocation	High — seed extractable from device	Medium — PIN + private key on device	None — device-independent	Variable — IdP session state
NTP	Yes —	Yes —	No	No	No

Feature	PKTBA (This Protocol)	TOTP (RFC 6238)	FIDO2 / WebAuthn / Passkeys	Magic Links	SAML / SSO
dependency	critical	important			
Network required at auth time	Yes (server contact) / No (token derivation)	No (token derivation only)	Yes (RP challenge)	Yes (email channel)	Yes (IdP redirect)
Quantum resistance posture	Good (SHA-3 / BLAKE3 variants available)	Good (HMAC-SHA-256)	Vulnerable (RSA / ECDSA keys)	Good (URL entropy)	Vulnerable (RSA / ECDSA signatures)
Server-side secret to protect	None (method array signing key only — public-metadata signer)	Per-user seed database	None (public key only)	Temporary nonce store	IdP credential database
Algorithm agility	Yes — method array rotation	Limited (RFC-defined)	Yes — algorithm negotiation	N/A	Limited
Implementation complexity	Medium — NTP hardening required	Low	High — WebAuthn API, platform integration	Low	High — IdP federation, SAML XML

10.1 vs. TOTP (RFC 6238)

TOTP requires a per-user seed — a shared symmetric secret that must be stored in the authenticator application and in the server's credential database. This seed is typically provisioned via a QR code scan (transmitting the seed over the user's camera), and the server stores it persistently. A breach of the server's seed database immediately compromises the future OTPs of every user, regardless of whether they are currently authenticated. This is not a theoretical risk: large-scale TOTP seed database breaches have been publicly reported from major identity providers.

PKTBA has no per-user seed. There is no database of per-user secrets to breach. The server stores only user IDs and the ephemeral replay ledger. A complete breach of the server's non-secret data provides an attacker with zero authentication capability.

10.2 vs. FIDO2/WebAuthn/Passkeys

FIDO2 passkeys represent the current industry gold standard for phishing-resistant passwordless authentication. The private key, stored in the device's Secure Enclave or TPM, is origin-bound — it can only be used to authenticate to the specific relying party domain it was registered with, providing inherent phishing resistance that PKTBA does not offer by default (see Section 12.2 for the domain-binding extension).

However, FIDO2 private keys represent a persistent credential on the device. Their security is bounded by the security of the platform's secure hardware and the unlock mechanism (PIN or biometric). On platforms where passkeys sync to cloud keystores (Apple iCloud Keychain, Google Password Manager), the private key — while encrypted — is resident in a cloud service that is itself a high-value attack target. PKTBA's device holds no such persistent credential; there is nothing to sync, nothing to exfiltrate, and nothing to attack at rest.

10.3 vs. Magic Links

Magic link authentication depends entirely on the security of the user's email channel. Email is inherently unencrypted at rest in many providers, subject to account compromise, and introduces a dependency on third-party email infrastructure latency. Magic links also have notoriously variable expiry policies. PKTBA is fully email-independent, operates in sub-second token computation time, and enforces strict 30-second token expiry. It requires no third-party communication channel and is entirely self-contained within the client-server pair.

10.4 vs. SAML/SSO

SAML-based SSO centralizes authentication trust in an Identity Provider (IdP) that holds user credentials. This creates a single high-value target whose compromise affects all connected relying parties. PKTBA is architecturally decentralized: any server that can synchronize its clock via NTP and hold a copy of the public method

array can independently validate authentication tokens. There is no federated trust dependency. This makes PKTBA particularly attractive for air-gapped enterprise environments or multi-cloud deployments where IdP federation introduces unacceptable latency or single-point-of-failure risk.

11. Implementation Guidance

11.1 Recommended Algorithm Suite

Component	Recommended Algorithm	Alternative	Notes
Token derivation (primary)	BLAKE3 keyed hash	HMAC-SHA-256	BLAKE3 is faster and provides better parallelism
Token derivation (legacy compat)	HMAC-SHA-256	HMAC-SHA-512	Widely supported in all platforms
Method array signing	Ed25519	ECDSA P-256	Ed25519 preferred for simplicity and speed
Transport security	TLS 1.3	—	Certificate pinning required for mobile clients
NTP security	NTS (RFC 8915) via Chrony 4.x	NTPsec with NTS	Both support NTS natively as of 2024
Session tokens (post-auth)	JWT (RS256 or ES256)	Opaque bearer tokens	Standard session management; not PKTBA-specific

11.2 Enrollment Flow

A defining advantage of PKTBA is its minimal enrollment ceremony. A user can begin authenticating with only the following steps:

18. Provide a UserID (username, email address, or opaque identifier).

19. Optionally: register a device identifier (hardware attestation certificate or fingerprint) for device binding.
20. Download the current method array (public metadata; no secret transmission).

There is no seed exchange, no QR code provisioning, no key generation ceremony, no app pairing, and no backup code issuance. Enrollment can be completed in under 5 seconds. This dramatically lowers the usability barrier for enterprise-scale rollout.

11.3 Deprovisioning

Revoking a user's authentication capability requires only removing their UserID from the active user store. Active sessions can be immediately invalidated by removing all session tokens associated with the UserID. There is no key revocation, no seed deletion, and no waiting for revocation propagation — because there is nothing cryptographic to revoke. Deprovisioning is instantaneous and complete.

11.4 Audit Logging

Each authentication attempt — successful or failed — should generate a structured audit log entry containing:

- **timestamp_utc:** ISO 8601 UTC timestamp of the authentication attempt
- **user_id:** The submitted UserID (hashed or pseudonymized for GDPR compliance if required)
- **client_ip:** Source IP address of the request
- **time_window_T:** The integer time window index evaluated
- **method_index_i:** The method array index used for validation
- **result:** SUCCESS | FAILURE | REPLAY | RATE_LIMITED | SYNC_REQUIRED
- **session_id:** Session identifier issued on success (null on failure)
- **delta_used:** Which tolerance offset (−1, 0, or +1) produced the match, if successful

11.5 Reference Implementation Sketch

The following pseudocode illustrates the server-side token validation logic. Production implementations must adapt this to their platform, language, and cryptographic library of choice.

```
# PKTBA Server-Side Token Validation — Reference Pseudocode # Version 1.0 | May 2026  WINDOW_SIZE = 30          # seconds per time window DELTA = 1 # tolerance windows (check T-1, T, T+1) TOKEN_BITS = 128          # minimum recommended token size  def validate_token(user_id: str, submitted_token: bytes) -> AuthResult:     """     Validates a PKTBA authentication token.     Returns AuthResult.SUCCESS or AuthResult.FAILURE.     """     # Step 1: Obtain NTS-synchronized UTC time     now_utc = get_nts_synchronized_utc_epoch()  # float: seconds since Unix epoch     # Step 2: Check per-user rate limit     if rate_limiter.is_blocked(user_id):         audit_log(user_id, "RATE_LIMITED")         return AuthResult.RATE_LIMITED     # Step 3: Evaluate all tolerance windows     for delta in range(-DELTA, DELTA + 1):     # [-1, 0, 1]         T = floor(now_utc / WINDOW_SIZE) + delta         method = METHOD_ARRAY[T % len(METHOD_ARRAY)]         # Step 4: Derive expected token (same function as client)         K_eph = H(encode(T) || encode(user_id))  # ephemeral pseudokey         expected = method.derive(T, user_id, K_eph)         # Step 5: Constant-time comparison (prevents timing oracle)         if constant_time_compare(submitted_token, expected):             # Step 6: Check replay ledger (atomic compare-and-set)             if replay_ledger.is_consumed(user_id, T):                 audit_log(user_id, "REPLAY", T=T)                 return AuthResult.REPLAY_DETECTED             # Step 7: Mark token consumed and issue session             replay_ledger.mark_consumed(user_id, T, ttl=WINDOW_SIZE * 3)             session = session_store.issue(user_id)             audit_log(user_id, "SUCCESS", T=T,
```

```
delta=delta,                    session_id=session.id)          return
AuthResult.SUCCESS      # Step 8: No match found — apply failure rate limiting
rate_limiter.record_failure(user_id)    audit_log(user_id, "FAILURE")    return
AuthResult.FAILURE
```

Critical Implementation Note

The `constant_time_compare()` function is mandatory. Do NOT use standard equality operators (`==`) for token comparison. Timing differences in early-exit string comparisons create an oracle that allows attackers to reconstruct the expected token byte-by-byte. Use `hmac.compare_digest()` in Python, `crypto.timingSafeEqual()` in Node.js, or equivalent constant-time comparison primitives in your language of choice.

12. Limitations and Future Work

12.1 NTP Dependency

PKTBA's most significant architectural dependency is reliable NTP synchronization. Unlike FIDO2, which operates correctly regardless of the system clock, PKTBA requires both client and server to agree on the current time to within the tolerance window. In environments where NTP is unavailable (air-gapped systems with degraded RTCs, satellite communication with high latency NTP), clock drift accumulates and eventually causes authentication failure.

Mitigation strategies for constrained environments include: GPS PPS disciplined local clocks, extended Δ values (at the cost of a wider replay window), and the `SYNC_REQUIRED`

error code mechanism that alerts the user to resynchronize before re-attempting authentication. These mitigations reduce but do not eliminate the NTP dependency.

12.2 Absence of Native Phishing Resistance

A notable security gap compared to FIDO2 is the absence of inherent phishing resistance. FIDO2 private keys are cryptographically bound to the origin URL of the relying party; they cannot be used to authenticate to a different domain even if the user is tricked into visiting a phishing site. PKTBA tokens, by default, are not origin-bound — a phishing page could relay a valid token to the legitimate server within the 30-second window.

The recommended mitigation is to include the relying party's domain as an additional input to the token derivation function: `τ = Derive(T, UserID, Domain, M[i])`. With this extension, a token computed for `auth.example.com` is cryptographically invalid at `auth.evil-phish.com`, even though both sites could trigger token computation. Implementations targeting high-assurance use cases should treat domain binding as mandatory, not optional.

12.3 The Method Array Signing Key

While PKTBA eliminates per-user secrets, it introduces a single system-wide cryptographic dependency: the method array signing key. This key's compromise would allow an attacker to distribute a malicious method array to clients, potentially causing predictable token derivation or algorithm downgrade. The attack requires the attacker to already be positioned to intercept method array distribution — a significant capability — but the risk is real and must be managed.

The signing key should be stored in a FIPS 140-3 Level 3 or higher Hardware Security Module. Key rotation should occur on a regular schedule (annually or after any suspected compromise). The public verification key should be distributed out-of-band at client installation time (similar to certificate pinning) to ensure clients can detect a compromised signing key.

12.4 Future Work

- **Formal Security Proof:** A rigorous security proof under the Universal Composability (UC) framework, modeling PKTBA as a secure authentication functionality against polynomial-time adversaries.
- **Decentralized Identity Integration:** Integration with W3C Decentralized Identifiers (DID v1.0) to allow UserIDs to be self-sovereign, removing the requirement for a centralized user registry.
- **Post-Quantum Method Suite Expansion:** Formal evaluation of SHA-3 and BLAKE3 variants against quantum adversaries using Grover's algorithm, and development of a post-quantum method array specification.
- **Zero-Knowledge Token Validation:** A zero-knowledge proof layer enabling server-side token validation without revealing the UserID to the verifier — enabling privacy-preserving authentication for anonymous credential use cases.
- **Standardization Submission:** Preparation of an Internet-Draft for submission to the IETF KITTEN working group or an analogous authentication standards body.
- **Cross-Domain Federation:** Extension of the method array versioning system to support multi-party federation, where multiple servers share a common signed method array for mutual authentication.

13. Conclusion

This paper has presented PKTBA — Private-Keyless, Passwordless, Time-Based Authentication — a novel identity verification protocol that achieves secrets-free authentication through the disciplined application of four mechanisms: precise NTP-synchronized time agreement, ephemeral token derivation from public inputs, rotating method array selection, and server-side replay prevention. The protocol has been designed, analyzed, and constrained by a single architectural principle: *there*

must be nothing persistent on either the client or the server that an attacker could steal and exploit for future authentication.

The core security insight of PKTBA is that time, when reliably synchronized and treated as an ephemeral shared reference rather than a credential, can serve as the coordinating factor in authentication — replacing the role played by passwords, seeds, and private keys in every prior authentication system. An adversary who fully compromises a PKTBA server's storage gains no authentication capability whatsoever. An adversary who steals a client device gains only a 30-second window of valid token computation, immediately terminable by server-side device binding revocation. An adversary who intercepts a token in transit gains a single-use value that expires in seconds.

The protocol's NTP dependency is its most significant structural constraint and the area that demands the greatest implementation discipline. The hardening measures described in Sections 4 and 5 — NTS (RFC 8915), NTPsec, multi-source NTP with Marzullo's algorithm, GPS-disciplined primary reference clocks for servers, and drift rate limiting — collectively transform the NTP layer from a potential liability into a robust trust anchor. Deployments that rigorously implement these measures inherit a time synchronization substrate that is resistant to all known practical attack vectors.

PKTBA does not position itself as a universal replacement for FIDO2 in all contexts. Where phishing resistance is the dominant requirement, FIDO2's origin binding provides capabilities that PKTBA does not offer natively (though domain binding as a derivation input, described in Section 12.2, substantially closes this gap). Where elimination of server-side secrets is the dominant requirement — as it is in enterprise breach response planning, regulatory compliance for credential storage, and zero-trust architecture design — PKTBA offers a compelling and practically deployable alternative.

The broader implication of this work is that the shared-secret paradigm — which underlies every major authentication system from passwords to TOTP seeds to SAML IdP credentials — is not a necessary architectural feature of identity verification. Time, universally accessible and continuously advancing, provides a coordination

mechanism that requires no secret to be shared, stored, or protected. Eliminating the secret eliminates the breach. That is the promise of PKTBA.

References

21. **[RFC6238]** M'Raihi, D., Machani, S., Pei, M., and J. Rydell. *TOTP: Time-Based One-Time Password Algorithm.* Internet Engineering Task Force (IETF), RFC 6238, May 2011. Available: https://www.rfc-editor.org/rfc/rfc6238
22. **[RFC4226]** M'Raihi, D., Bellare, M., Hoornaert, F., Naccache, D., and O. Ranen. *HOTP: An HMAC-Based One-Time Password Algorithm.* IETF, RFC 4226, December 2005. Available: https://www.rfc-editor.org/rfc/rfc4226
23. **[RFC5905]** Mills, D., Martin, J., Burbank, J., and W. Kasch. *Network Time Protocol Version 4: Protocol and Algorithms Specification.* IETF, RFC 5905, June 2010. Available: https://www.rfc-editor.org/rfc/rfc5905
24. **[RFC8915]** Franke, D., Sibold, D., Teichel, K., Dansarie, M., and R. Sundblad. *Network Time Security for the Network Time Protocol.* IETF, RFC 8915, September 2020. Available: https://www.rfc-editor.org/rfc/rfc8915
25. **[RFC5906]** Haberman, B. and D. Mills. *Network Time Protocol Version 4: Autokey Specification.* IETF, RFC 5906, June 2010. Available: https://www.rfc-editor.org/rfc/rfc5906
26. **[FIDO2]** FIDO Alliance. *FIDO2: Web Authentication (WebAuthn) — Overview and Specification.* FIDO Alliance, 2023. Available: https://fidoalliance.org/fido2/
27. **[NISTSP800-63B]** Grassi, P., Garcia, M., and J. Fenton. *NIST Special Publication 800-63B: Digital Identity Guidelines — Authentication and Lifecycle Management.* National Institute of Standards and Technology, 2017 (revised 2023). Available: https://pages.nist.gov/800-63-3/sp800-63b.html
28. **[FIDO2SEC]** Barbosa, M., Boldyreva, A., Chen, B., and B. Warinschi. *Provable Security Analysis of FIDO2.* In Advances in Cryptology – CRYPTO 2021. Lecture Notes in Computer Science, vol. 12827. Springer, 2021. DOI: 10.1007/978-3-030-84252-9_5

29. **[NTPSEC]** NTPsec Project. *NTPsec: A Secure, Hardened, and Improved Implementation of NTP.* The NTPsec Project, 2024. Available: https://www.ntpsec.org/

30. **[DBIR2024]** Verizon. *2024 Data Breach Investigations Report (DBIR).* Verizon Business, 2024. Available: https://www.verizon.com/business/resources/reports/dbir/

31. **[WEBAUTHN3]** Balfanz, D., et al. *Web Authentication: An API for Accessing Public Key Credentials — Level 3.* W3C Working Draft, World Wide Web Consortium, 2023. Available: https://www.w3.org/TR/webauthn-3/

32. **[BLAKE3]** O'Connor, J., Aumasson, J.-P., Neves, S., and Z. Wilcox-O'Hearn. *BLAKE3 — One Function, Fast Everywhere.* 2020. Available: https://github.com/BLAKE3-team/BLAKE3-specs/

33. **[MARZULLO]** Marzullo, K., and S. Owicki. *Maintaining the Time in a Distributed System.* ACM SIGOPS Operating Systems Review, 19(3): 44–54, 1985.
