



KeyZula

— SECURITY WHITEPAPER

How KeyZula protects your vault

Architecture, cryptography, account protection and the limits of what our servers can see.

VERSION

1.1

DATE

30 September 2026

PUBLISHER

Netlen İnternet Hizmetleri Ltd. Şti.

Contents

1	Introduction	4
2	Threat model and security goals	4
2.1	Security goals	4
2.2	Adversaries considered	4
2.3	Out of scope	5
3	Cryptographic primitives	5
4	Account creation and login	6
4.1	Key derivation	6
4.2	Registration	6
4.3	Login and verifier storage	6
4.4	Sessions	6
4.5	Two-factor authentication	7
4.6	Passkey sign-in	7
4.7	New-device notices and security e-mails	7
4.8	Rate limiting	7
4.9	Unlocking without the master password	8
4.10	Password change and account deletion	8
5	Vault encryption	8
5.1	Key hierarchy	8
5.2	Items	9
5.3	Associated data	9
5.4	Metadata stored in the clear	9
6	Sharing and organizations	10
6.1	Organizations and collections	10
6.2	Adding members and fingerprint verification	10
6.3	Permissions	10
6.4	Removing access	10
6.5	Policies	11
7	SSO, trusted devices and admin recovery	11

7.1	Single sign-on	11
7.2	Accounts without a master password	11
8	Emergency access	12
9	Send	12
10	Attachments and files	12
11	Service accounts and the command-line tool	13
11.1	Service accounts	13
11.2	Command-line tool	13
12	Browser extension	13
13	Mobile apps	14
14	Web vault	15
15	What the server can and cannot see	15
16	Transport and infrastructure	16
17	Access logs and audit events	17
18	Design considerations	17
19	Responsible disclosure	18

1 Introduction

KeyZula is an end-to-end encrypted vault for passwords, passkeys, two-factor authentication (TOTP) codes, SSH keys, documents and notes, for individuals and organizations. It is developed and operated by Netlen İnternet Hizmetleri Ltd. Şti., İstanbul. KeyZula is available as a web vault (app.keyzula.com), browser extensions, iOS/iPadOS and Android apps, and a command-line tool.

This document describes how KeyZula protects vault data: how keys are derived and stored, how data is encrypted and shared, how accounts and sessions are protected, what the KeyZula server can and cannot see, and the design considerations behind these choices. It reflects the implementation as of the date on the cover.

The client-side encryption library used by all KeyZula clients is open source under the Apache 2.0 license at github.com/KeyZula/keyzula-crypto. The server, web vault, extensions and apps are closed source; this document describes their security-relevant behavior.

2 Threat model and security goals

2.1 Security goals

- **Confidentiality against the server.** Vault contents (item names, usernames, passwords, URLs, notes, custom fields, TOTP secrets, passkeys, SSH keys, attachments and their file names) are encrypted on the user's device. The server stores and synchronizes ciphertext and cannot decrypt it.
- **Integrity and context binding.** All ciphertext is authenticated. Each ciphertext is bound to its purpose and record identifier, so the server cannot swap encrypted values between items, fields or attachments without detection.
- **No offline attack from server data alone** beyond guessing the master password, which is deliberately made expensive with Argon2id.
- **Account protection** against credential stuffing and online guessing through a server-side verifier, rate limits, two-factor authentication and notifications.

2.2 Adversaries considered

- An attacker who obtains a copy of the database, backups or attachment storage.
- A malicious or compromised server operator, including one who returns manipulated data (for example weak KDF parameters or swapped ciphertexts).
- A network attacker between client and server.

- Other users, including members and administrators of a shared organization, acting beyond their permissions.

2.3 Out of scope

- A compromised end-user device (malware, keyloggers, a malicious browser extension with access to page content) while the vault is unlocked.
- A weak or reused master password chosen by the user.
- Availability: an operator can refuse service or delete data; users can export their vault.
- Malicious client code. The web vault is delivered by the KeyZula server on every visit; an attacker who controls that server could serve modified code. Native apps and store-distributed extensions are not affected by this in the same way (see Design considerations).

3 Cryptographic primitives

Clients use `libsodium` (`libsodium-wrappers-sumo` , WebAssembly) and `@noble/hashes` . All random values come from `libsodium`'s CSPRNG.

PURPOSE	ALGORITHM AND PARAMETERS
Master password key derivation	Argon2id (v1.3); default 3 iterations, 64 MiB memory; 16-byte random salt per account
Subkey derivation	HKDF-SHA256 with distinct labels (for example <code>vault/v1/auth-hash</code> , <code>vault/v1/key-encryption-key</code>)
Symmetric encryption	XChaCha20-Poly1305 (IETF), 256-bit keys, random 192-bit nonce per encryption, associated data (AAD) bound to context
Public-key encryption	X25519 sealed boxes (<code>crypto_box_seal</code>)
Public key fingerprints	BLAKE2b-256 over a label, the account e-mail and the public key; displayed as 6 groups of 5 digits
Server-side verifiers	Argon2id (2 iterations, 19 MiB) for login and PIN verifiers; SHA-256 for session and service tokens

Symmetric ciphertexts are serialized as `v1.<nonce>.<ciphertext>` in base64url. The version prefix allows future algorithm changes.

4 Account creation and login

4.1 Key derivation

The master password is normalized (Unicode NFKC) and processed with Argon2id using the account's salt and cost parameters to produce a 256-bit master key. Two independent keys are derived from it with HKDF-SHA256:

- **Authentication hash** (`vault/v1/auth-hash`): sent to the server to prove knowledge of the password.
- **Key-encryption key (KEK)** (`vault/v1/key-encryption-key`): never leaves the device; it decrypts the user key.

The master key is wiped from memory after derivation. Clients enforce bounds on KDF parameters received from the server (Argon2id only; 2–10 iterations; 19 MiB – 1 GiB memory; 16-byte salt), so a malicious server cannot downgrade the KDF or exhaust the client. The server validates the same bounds.

4.2 Registration

On registration the client generates, on the device:

- a random 256-bit **user key**, encrypted with the KEK (`protectedUserKey`);
- an **X25519 key pair**; the private key is encrypted with the user key (`protectedPrivateKey`), the public key is stored in the clear.

The server receives only the KDF parameters, the authentication hash, the public key and these two ciphertexts. A verification e-mail is sent after registration.

4.3 Login and verifier storage

- The client first requests the account's KDF parameters (prelogin). For unknown e-mail addresses the server returns deterministic parameters derived with HMAC-SHA256 from a server secret, so prelogin does not reveal whether an address is registered.
- The server never stores the authentication hash itself. It stores an Argon2id hash of it (`libsodium crypto_pwhash_str`). A copy of the database therefore cannot be used to log in.
- When the account does not exist the server still performs a verification against a dummy verifier, and returns the same error for an unknown address and a wrong password.
- After successful authentication the client decrypts the user key with its KEK and the private key with the user key. Decrypted keys are kept in memory only.

4.4 Sessions

- A session token is 32 random bytes (`vs_` prefix). The server stores only its SHA-256 hash.

- Tokens are sent as `Authorization: Bearer` headers; no cookies are used for authentication. The server's request logger redacts the authorization header.
- Sessions expire after a fixed lifetime (30 days by default). Users can list their sessions (device, IP address, last activity) and revoke any of them. Changing the master password revokes all other sessions.

4.5 Two-factor authentication

- **Authenticator app (TOTP):** SHA-1, 6 digits, 30-second period, 160-bit secret; ± 1 time step accepted; a used time step cannot be reused (also under concurrent requests). TOTP secrets are encrypted at rest with AES-256-GCM using a key derived from the server secret, because the server must be able to verify codes.
- **Security keys (WebAuthn):** up to 20 keys per account; single-use challenges valid for 5 minutes.
- **Recovery code:** one 160-bit code, stored as a keyed HMAC-SHA256. Using it disables two-factor authentication, is logged and triggers an e-mail.
- Enabling or changing two-factor settings requires re-authentication. After 10 failed second-factor attempts within 15 minutes, further attempts are refused.

4.6 Passkey sign-in

Users can register passkeys for signing in to KeyZula. Registration requires a discoverable credential with user verification. At sign-in, the WebAuthn PRF extension is evaluated with a fixed input on the authenticator; the PRF output is processed with HKDF (`vault/v1/passkey-key`) into a key that wraps the user key. The PRF output never leaves the device; the server stores only the wrapped user key and the credential's public key. Because the passkey itself requires possession and user verification, account two-factor authentication is not requested again for passkey sign-in; organization SSO requirements still apply.

4.7 New-device notices and security e-mails

Each client keeps a random device identifier; the server stores only its SHA-256 hash. When a login (password, passkey or SSO) comes from a device not seen before on an account that already has known devices, the user receives an e-mail with the device name, IP address, time and browser. E-mails are also sent for password changes, e-mail address changes (to the old address), enabling or disabling two-factor authentication and account deletion.

4.8 Rate limiting

- Global limit per IP address (300 requests per minute) and stricter per-IP limits on authentication endpoints (10 per minute by default).
- Per-account limit on failed master-password attempts (20 per 15 minutes by default), independent of the source IP.
- Registration, e-mail sending, invitations, Send access and service-account authentication have their own limits.

4.9 Unlocking without the master password

Clients can re-open a locked vault with a PIN, biometrics or, in the web vault, across page reloads. For PIN and reload unlock, the user key is wrapped with a key derived by HKDF from a local secret and a random 32-byte **server share** (vault/v1/unlock-key). The server releases the share only to the session that created it:

- **PIN:** the PIN is processed with Argon2id on the device; the server stores an Argon2id hash of the resulting verifier and deletes the share after 5 wrong PINs, after which the master password is required. A stolen copy of the locally stored wrapped key therefore cannot be brute-forced offline.
- **Reload (web):** the share expires after a configurable period of inactivity (at most 12 hours) and is deleted when the session ends.
- **Biometrics (mobile):** the user key is stored in the iOS Keychain / Android Keystore-backed secure storage with user-authentication required and "this device only" accessibility.

Shares are deleted together with their session (logout, revocation, password change).

4.10 Password change and account deletion

Changing the master password generates a new salt and re-wraps the unchanged user key with the new KEK; vault items are not re-encrypted. Account deletion requires re-authentication and removes the account, its sessions, devices, credentials and personal vault data.

5 Vault encryption

5.1 Key hierarchy

KEY	PROTECTED BY	ENCRYPTS
Master key	Derived from the master password, never stored	— (source of KEK and authentication hash)
User key	KEK (and optionally passkey, PIN, device or recovery wrappings)	Private key, personal item keys, folder names, Send keys
X25519 private key	User key	Opens collection keys, emergency-access and device-approval keys sealed to the user
Collection key	Sealed to each authorized member's public key	Item keys of shared items, collection names
Item key (per item)	User key or collection key	Item data, attachment keys

KEY	PROTECTED BY	ENCRYPTS
Attachment key (per file)	Item key	File contents, file name and type

5.2 Items

Every item has its own random 256-bit key. All item content — type, name, username, password, URLs, notes, custom fields, favorite flag, TOTP secret, passkey, SSH key and password history — is serialized and encrypted as a single ciphertext. Moving an item between a personal vault and a collection re-wraps only the item key; the content is not re-encrypted.

5.3 Associated data

Every encryption binds a context string as associated data. Decryption with a different context fails authentication, which prevents the server from moving ciphertext between records or fields.

VALUE	ASSOCIATED DATA
User key (wrapped by KEK)	<code>vault/v1/user-key</code>
Private key	<code>vault/v1/private-key</code>
Item key / item data	<code>vault/v1/item-key:<id></code> , <code>vault/v1/item:<id></code>
Folder / collection name	<code>vault/v1/folder:<id></code> , <code>vault/v1/collection:<id></code>
Attachment key / metadata	<code>vault/v1/attachment-key:<id></code> , <code>vault/v1/attachment-meta:<id></code>
Send key / content	<code>vault/v1/send-key:<id></code> , <code>vault/v1/send:<id></code>
Alternative user-key wrappings	<code>vault/v1/pin-user-key</code> , <code>vault/v1/reload-user-key</code> , <code>vault/v1/passkey-user-key</code> , <code>vault/v1/device-user-key</code>
Verified member key (pin)	<code>vault/v1/key-pin:<member id></code>

5.4 Metadata stored in the clear

To synchronize, the server keeps per item: its identifier, owner (user or collection), folder identifier, revision counter, creation, update and deletion (trash) timestamps, and the size of the ciphertext. Deleted items stay in the trash until the user deletes them permanently. The server keeps no item version history; password history is part of the encrypted item.

6 Sharing and organizations

6.1 Organizations and collections

Organizations group members into departments and share items through collections. Each collection has a random key. For every member with access, the collection key is sealed to that member's X25519 public key and stored on the server; the member opens it with their private key. Collection names are encrypted with the collection key.

Organization names, department names, member e-mail addresses, roles and permissions are stored in the clear, because the server enforces access control on them.

6.2 Adding members and fingerprint verification

Members are invited by e-mail, accept with a verified e-mail address, and are then confirmed by an owner or administrator. Only confirmed members receive collection keys; the administrator's client seals them automatically.

Because the server distributes public keys, it could in principle present a different public key for a member. To detect this, the confirmation dialog shows the member's public key fingerprint, and each member can see their own fingerprint in their account settings. Administrators compare the two over a separate channel before confirming.

The key an administrator confirms is then **pinned**: the administrator's client encrypts it with the administrator's own user key, bound to the membership (`vault/v1/key-pin:<member id>`), and stores the result on the server, which cannot create or change a pin without detection. Before sealing a collection key or the organization recovery key to a member, the administrator's client compares the public key supplied by the server with its pin. If they differ, or if this administrator has not verified the member, nothing is sealed and the administrator is shown which members need their fingerprint compared (again). Each administrator keeps their own pins; an administrator's own key is known to their client and needs no pin. Account public keys cannot be changed after registration, so a differing key is not expected in normal operation.

6.3 Permissions

Collection permissions are *manage*, *edit*, *view* and *view, password hidden*. Owners and administrators can manage all collections. The server enforces who may read, create, modify and delete items. The *view, password hidden* permission is a usability control: members can use the item (for example for autofill) but clients do not display, copy or export its secrets. These members hold the collection key, so the permission is not a cryptographic boundary; a modified client could reveal the secrets. Attachment downloads are refused by the server for this permission.

6.4 Removing access

When a member is removed, or loses access to a collection, the server deletes the sealed collection keys for that member. Collection keys are not rotated. A removed member who kept a copy of a

collection key and of ciphertext could still decrypt data encrypted under that key before their removal.

6.5 Policies

Organizations can disable vault export for members. Policies are applied by the clients.

7 SSO, trusted devices and admin recovery

7.1 Single sign-on

Organizations can connect an OpenID Connect identity provider. The KeyZula server performs the authorization-code flow with PKCE, state and nonce; the client redeems a one-time ticket with its own PKCE verifier. The identity provider's e-mail domain must match a domain the organization has verified with a DNS TXT record, and e-mail addresses reported as unverified are rejected. The provider's client secret is encrypted at rest with a key derived from the server secret. SAML is not supported.

The identity provider authenticates the user; it never receives vault keys. Organizations can require SSO for members; owners are exempt from this requirement.

7.2 Accounts without a master password

Users who join through SSO can use KeyZula without a master password. Their user key and key pair are generated on the device. The user key is then made available on other devices in three ways:

- **Trusted device.** A random 256-bit device key stays on the device. On the web it is kept in browser storage encrypted with a non-extractable AES-GCM key held in IndexedDB, so the stored value alone does not reveal it; on mobile it is kept in secure storage. The server stores the user key wrapped with this device key. Users can list and remove trusted devices.
- **Device approval.** A new device generates an ephemeral X25519 key pair and publishes the public key. Another signed-in device of the same user, or an organization administrator (see below), seals the user key to it. Both sides display a verification number (fingerprint of the ephemeral key) to compare. Requests expire after 15 minutes and the result is delivered only to the requesting session.
- **Admin recovery.** The organization has its own X25519 key pair; its private key is sealed to each owner and administrator, only to keys pinned as described under fingerprint verification. Members' user keys are sealed to the organization public key. An administrator can therefore approve a member's new device and, in doing so, gains the ability to decrypt that member's vault key.

Admin recovery is disclosed and requires the user's consent. When a new account is enrolled through SSO, the enrollment screen states that organization administrators can open the vault and requires explicit acknowledgement, which the server also checks. When an existing account signs in through an organization's SSO and has no recovery copy for that organization yet, the same notice is shown

before anything is shared: the recovery copy is created and uploaded only after the user accepts, and the server rejects uploads without the acknowledgement. If the user declines, no copy is shared; administrators then cannot approve the user's new devices or restore access, and the notice is shown again at the next SSO sign-in.

For sensitive actions, accounts without a master password re-authenticate with a key proof: the server seals a random challenge to the account's public key and the client answers with the decrypted value (single use, 5-minute lifetime).

8 Emergency access

Users can nominate trusted contacts (up to 50). A contact is invited by e-mail, accepts with a verified address, and is confirmed by the grantor. On confirmation the grantor's client seals the grantor's user key to the contact's public key; the confirmation screen shows the fingerprint of that same key for out-of-band comparison, and the key is sealed only at that moment.

The contact can request access at any time. The grantor is notified by e-mail and can approve or reject the request; if the grantor does nothing, access opens after the waiting period chosen by the grantor (1, 2, 7, 14 or 30 days), enforced by the server. Access is view-only and covers the grantor's personal vault items and attachments, not organization collections. Once the sealed key has been released, later revocation does not rotate the grantor's user key.

9 Send

Send creates an expiring, end-to-end encrypted link for text. The content is encrypted with a random link key. The link has the form `https://app.keyzula.com/send/<id>#<key>`; the key is in the URL fragment, which browsers do not send to the server. A copy of the link key wrapped with the owner's user key lets the owner copy the link again later.

- Expiry: at most 31 days. Optional maximum number of views (1–1000), enforced atomically.
- Optional password: the client sends a BLAKE2b verifier keyed with the link key; the server stores an Argon2id hash of it and limits wrong attempts. The password gates access on the server; it is not an additional encryption layer.
- Unavailable, expired and non-existent Sends return the same response.

10 Attachments and files

Every attachment has its own random key, wrapped with the item key. File name and MIME type are encrypted. File contents use the **KZA1** chunked format:

```
file = "KZA1" || chunk_0 || ... || chunk_n
chunk = nonce (24 bytes) || XChaCha20-Poly1305(plaintext ≤ 1 MiB) incl. 16-byte tag
AAD = "vault/v1/attachment:<id>" || 0x00 || u32be(index) || u8(isFinal)
```

Binding the chunk index and a final-chunk flag prevents reordering, truncation and mixing chunks between files. Files are encrypted before upload and stored on the server's disk under random identifiers. The server checks that the uploaded size matches the declared ciphertext size for quota enforcement; the ciphertext size, and therefore the approximate file size, is visible to the server. Default limits are 25 MB per file and 100 MB per account.

11 Service accounts and the command-line tool

11.1 Service accounts

Service accounts give servers, CI jobs and automated agents read-only access to selected collections. An access token is created in the administrator's browser and shown once:

- The 256-bit token secret is split with HKDF into an **auth key** and a **wrap key**. Clients send only the auth key (`kzsa_auth_...`); the server stores its SHA-256 hash and rejects raw tokens.
- Each token has its own X25519 key pair. Its private key is encrypted with the wrap key; collection keys are sealed to its public key. The server never sees the token secret, wrap key, private key or collection keys.
- Tokens can expire and can be revoked individually; revocation deletes the verifier, the encrypted private key and the sealed collection keys. Every read is recorded in the organization's audit log.

11.2 Command-line tool

The CLI (`keyzula`) stores the server URL, e-mail, session token and the encrypted key material in a configuration file readable only by the user (mode 0600). The master password and decrypted keys are held only in the memory of a local agent process. Service tokens are read from an environment variable and are not persisted.

The built-in SSH agent serves SSH keys from the vault over a user-only socket. Private keys stay in memory; optional per-signature confirmation and idle locking are available; requests to add or remove keys through the agent protocol are refused.

12 Browser extension

- **Key storage.** Decrypted keys are kept in the background service worker and in the browser's session storage area, which is held in memory and cleared when the browser closes. Local

extension storage holds the session token, encrypted account keys, settings and an encrypted copy of the vault.

- **Auto-lock.** Users can lock after a period of inactivity or when the operating system locks; by default the vault locks when the browser closes.
- **Autofill matching.** By default an item matches pages on the same registrable domain (eTLD+1, using the Public Suffix List including private suffixes, so `a.github.io` does not match `b.github.io`). Per-URL rules allow exact host, prefix, exact URL or never. Items whose saved URLs all use `https` are not filled into plain `http` pages.
- **User action required.** The extension never fills on page load. Filling requires a click in the popup, the keyboard shortcut, or a click in the inline menu, which runs in a closed shadow DOM and accepts only trusted user events. Each frame re-checks its own URL against the item before filling, and page origins are taken from the browser, not from page messages.
- **Embedded frames.** Inline suggestions and filling are offered only in the top-level page and in frames on the same site (registrable domain) as the top-level page, for example a `login.example.com` frame inside `www.example.com`. The frame's and the page's addresses are taken from the browser. Frames embedded from a different site get no inline menu and are not filled; a fill requested from the popup for such a frame is refused with an explanation.
- **Passkeys.** The extension can act as a passkey provider. It validates the relying-party ID against the requesting origin (HTTPS only; the RP ID must be the host or a parent domain and cannot be a public suffix) and requires a trusted click in its overlay. Passkey requests are handled only in the top-level page; in embedded frames the browser's own passkey flow is used.
- **Clipboard.** Copied secrets are cleared after a configurable time (60 seconds by default).
- **Signing in with a passkey.** The extension opens the web vault with an ephemeral public key in the URL fragment; the web vault seals the user key (or PRF output) to that key and returns it through a channel restricted to the KeyZula origin. Requests are single use and expire.

13 Mobile apps

- **Key storage.** The session token and unlock data are stored in the iOS Keychain / Android Keystore-backed secure storage with "when unlocked, this device only" accessibility. Biometric unlock requires user authentication for every access. Android backups are disabled.
- **Auto-lock.** The app locks after a configurable time in the background (5 minutes by default) and hides its content in the app switcher.
- **iOS AutoFill.** The AutoFill extension reads a cache that is encrypted to a device key held in the Keychain with user-presence protection; every fill requires Face ID, Touch ID or the device passcode. Items with the *view, password hidden* permission are not included.
- **AutoFill matching.** On iOS and Android, websites are matched by registrable domain using the Public Suffix List, as in the browser extension. The app computes each item's match keys (the registrable domain, or the host name for host-, prefix- and exact-URL rules, since the AutoFill services receive only the site's host) and the AutoFill service derives the same keys for the site being filled from a copy of the list bundled with the app. On Android, a website domain reported in

a fill request is used only when the requesting app is a known browser whose signing certificate matches the published list of browsers; other apps are matched by their package name.

- **Android Autofill.** Suggestions show only names and usernames. Passwords and codes are decrypted only after biometric or device-credential authentication, using a Keystore key that requires authentication for every use.
- **Clipboard.** On iOS copied values are local-only (not shared via Universal Clipboard) and expire; on Android they are marked sensitive and cleared after the timeout.

14 Web vault

- Decrypted keys are held in memory only and wiped when the vault locks or the user signs out. Browser storage contains the session token, encrypted account keys, if enabled the PIN- or reload-wrapped user key described above, and on trusted SSO devices the encrypted device key.
- The vault locks after inactivity (15 minutes by default). Copied secrets are cleared from the clipboard after 60 seconds by default.
- The web vault is served with a strict Content Security Policy (scripts only from its own origin; no inline scripts; framing forbidden) and contains no analytics or advertising scripts. Key derivation runs in a Web Worker.
- Breach checks use the Have I Been Pwned range API with k-anonymity: only the first five characters of the password's SHA-1 hash are sent, with response padding. Checks are started by the user.

15 What the server can and cannot see

DATA	SERVER ACCESS
Master password, master key, KEK	Never received
User key, private key, collection keys, item keys	Encrypted only
Item contents: names, usernames, passwords, URLs, notes, fields, TOTP secrets, passkeys, SSH keys, item type, favorites	Encrypted only
Folder and collection names	Encrypted only
Attachment contents, file names and types	Encrypted only
Send contents and link keys	Encrypted only (link key never sent)

DATA	SERVER ACCESS
Passkey PRF outputs, trusted-device keys, service token secrets	Never received
Verified member keys pinned by administrators	Encrypted only (with the administrator's user key)
Authentication hash, PIN verifier	Stored as Argon2id hashes
E-mail address, name, account settings	Visible
Item identifiers, counts, timestamps, folder assignment, ciphertext sizes	Visible
Organization and department names, members, roles, permissions	Visible
Sessions and devices (device name, IP address, user agent), access and audit events	Visible
TOTP secret for the KeyZula account's own two-factor authentication	Encrypted with a server key (needed to verify codes)
Website hostnames when site icons are enabled	Visible in icon requests (see below)

Site icons are fetched by the KeyZula server from the website itself, not from a third-party icon service. Clients request icons by hostname, so the server can observe which hostnames a client asks for; the icon endpoint does not log requests. Icons can be turned off in settings.

16 Transport and infrastructure

- **Hosting.** KeyZula's servers are located in Türkiye and operated by Netlen İnternet Hizmetleri Ltd. Şti.
- **TLS.** All traffic uses HTTPS with automatically managed certificates (TLS 1.2 or later; HTTP/3 supported). HTTP Strict Transport Security is enabled with a one-year lifetime including subdomains.
- **Headers.** The web vault and API are served with a Content Security Policy (`default-src 'self'` , `script-src 'self' 'wasm-unsafe-eval'` , `frame-ancestors 'none'` , `object-src 'none'` , `base-uri 'none'` , `form-action 'self'`) and a restrictive Permissions-Policy. The API uses bearer tokens and does not allow credentialed cross-origin requests.
- **Isolation.** The application, database and reverse proxy run as separate containers; the database is not exposed to the network.

- **Server secret.** A single server secret is the root of keys used for server-side functions (encryption of TOTP secrets and SSO client secrets, signed e-mail links, prelogin parameters for unknown addresses). It is never used for vault data, which the server cannot decrypt.
- **Backups.** The database is backed up daily. Backups are kept for 14 days and then deleted automatically, both on the server and in off-site copies. Off-site copies are encrypted (AES-256, GnuPG) before upload; attachments deleted from the service are removed from them within 14 days. Backups contain vault data only in encrypted form.
- **Request hygiene.** Request body sizes are limited, requests containing NUL characters are rejected and database errors are not returned to clients.

17 Access logs and audit events

KeyZula records security events (logins, failed logins, password and e-mail changes, two-factor changes, passkey changes, device trust and approvals, organization, member, collection and policy changes, service-account activity) and item access events (viewed, password revealed, copied, autofilled, passkey used, SSH key used, attachment downloaded, read via CLI or service account).

- Clients report access events with the item identifier and action only, never item names or contents. The server adds the IP address, client type and device name, and checks that the user can access the item.
- Access events are kept for 365 days.
- Events on personal vault items are visible only to the vault owner; organization administrators never see them. Events on shared items are visible to organization owners and administrators; other members see only their own.

18 Design considerations

The following trade-offs follow from KeyZula's design and are stated here so that they can be taken into account when deciding how to use it.

- **Closed source.** Only the encryption library is open source. The server, web vault, extensions and apps are closed source; this document describes their behavior.
- **Web code delivery.** As with any web application, the web vault's code is delivered by the server each time it loads. A compromised server could serve modified code to web users. Native apps and extensions installed from app stores do not load their code from KeyZula's servers.
- **Metadata.** The metadata listed above (account information, item counts and timestamps, organization structure, IP addresses, access events) is visible to the server.
- **Key rotation on revocation.** Removing a member, revoking a service-account token or ending emergency access deletes the server-side key copies but does not rotate collection keys or user keys.

- **Client-enforced controls.** The *view, password hidden* permission and organization policies are enforced by KeyZula clients, not cryptographically.
- **Fingerprint comparison.** Clients pin and enforce the key an administrator confirms; whether the fingerprint was actually compared with the person at confirmation depends on the administrator.
- **Admin recovery.** In organizations using SSO with account recovery, administrators can gain access to members' vault keys by design; this is disclosed to users, and the recovery copy is shared only after they accept.
- **Master password.** KeyZula cannot reset a forgotten master password. Accounts without other unlock methods lose access to their vault.

19 Responsible disclosure

We welcome security reports. Please e-mail info@keyzula.com with the subject "**Güvenlik**" (Turkish for "Security"), describing the issue, how to reproduce it and its potential impact. Please give us reasonable time to fix the issue before publishing details, limit testing to your own account, and do not access other users' data, disrupt the service or use social engineering. Contact details are also published at <https://keyzula.com/.well-known/security.txt>.

Reports about the open-source library can also be sent to the same address; see [SECURITY.md](#) in the repository.