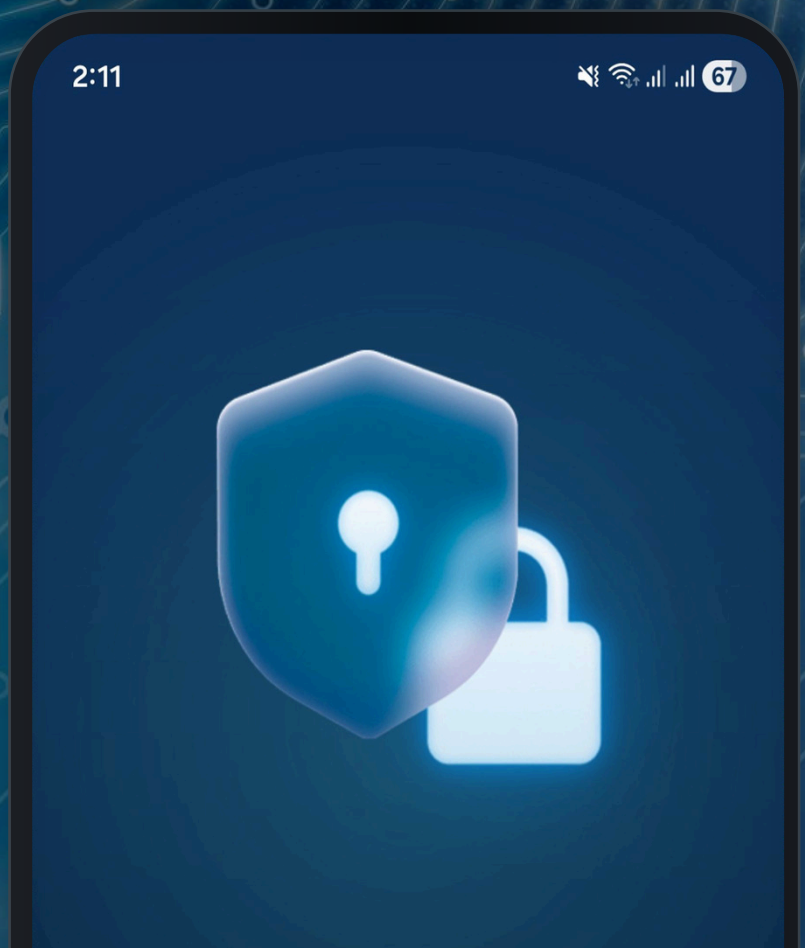
HIDEEZ

WHITEPAPER

Hideez Mobile Authenticator Security Architecture



www.hideez.com

Abstract

The purpose of this document is to explain how Hideez Authenticator is designed and manufactured, what organizational and technical measures have been taken to ensure information security, and what are the possible risks associated with them.

The document is intended for information security specialists and is meant to be a source of information to help them decide on the appropriateness of using Hideez Authenticator within a specific company. It does not provide a complete description of the functionality and benefits for users.

Document	Hideez Mobile Authenticator — Security Architecture
Version	2.0
Date	August 2026
Approved by	Denys Zaliznyak, CTO, Hideez Group

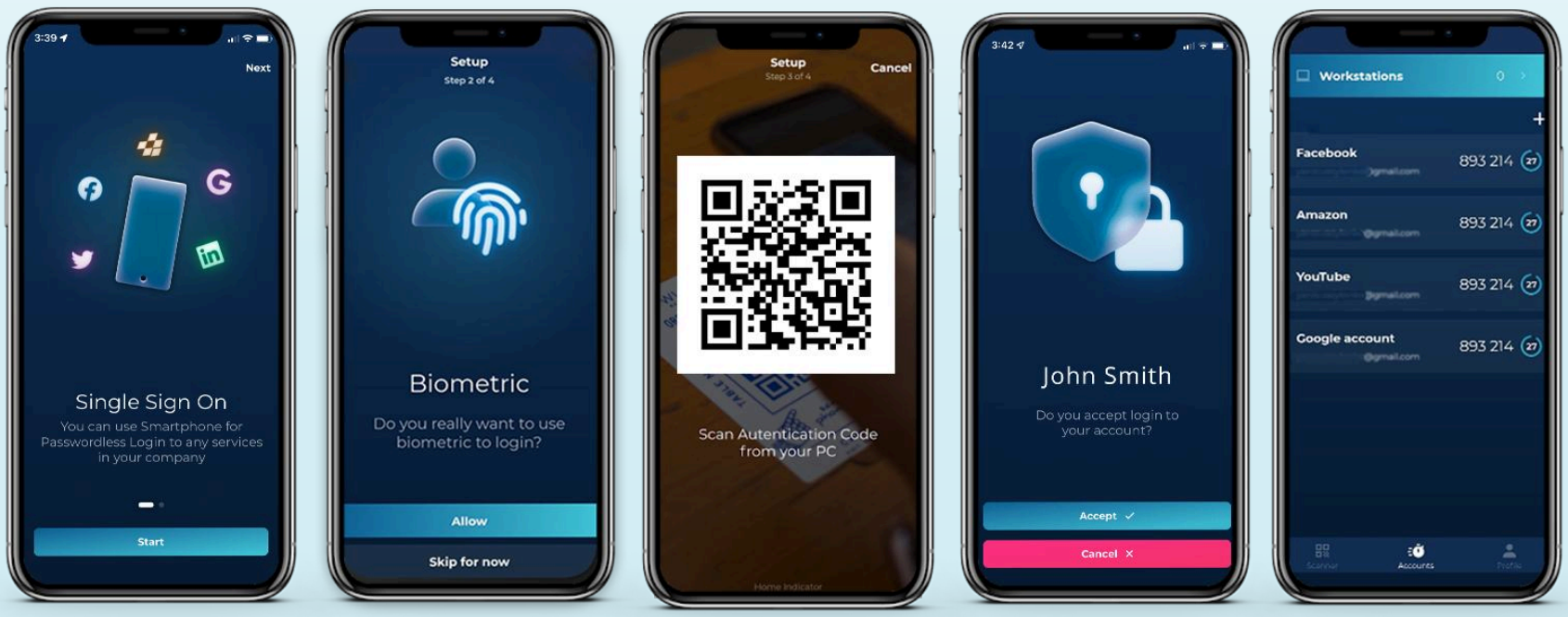


Table of contents

Terms and definitions

AD	Microsoft Active Directory server (on premise)
AES	Advanced Encryption Standard — symmetric encryption algorithm
Bluetooth LE, BLE	Bluetooth Low Energy, a subset of the Bluetooth protocol that enables communication between devices in a low power mode
Bond	(bond file) — Bluetooth pairing information of the device including MAC address and AES-128 encryption key
Device Master Key	AES-128 encryption key to authorize and encrypt user data on the device
ECDH	Elliptic Curve Diffie-Hellman
FIDO	Fast IDentity Online, the passwordless authentication standard and the eponymous alliance that develops and promotes this standard
HES	Hideez Enterprise Server
MAC address	a unique identifier assigned to each unit of active equipment or some of their interfaces in computer networks
MITM	Man-in-the-middle attack
OTP	one-time password or time-based one-time password (TOTP) based on RFC 6238 specification
OTP Secret	a secret key for generating one-time passwords
RSA	Rivest–Shamir–Adleman — asymmetric encryption algorithm
TPM	Trusted Platform Module

Application security overview

Hideez Authenticator is a free app for Android and iOS that turns a smartphone into a reliable key for logging into services and workstations.

Hideez Authenticator is part of Hideez Workforce Identity and is used in corporate projects. This paper describes the security mechanisms of the mobile application and the channels it uses.

MINIMUM IOS	MINIMUM ANDROID
13	8

Security rests on eight pillars

01 Local user authentication

02 Internal application data encryption and key management

03 Enrollment process

04 Data transfer via the server

05 Bluetooth communication security

06 Workstation login and unlock with Hideez Desktop

07 QR code based authentication on HES

08 Root check and jailbreak detection

SECURITY PILLAR 01

Local user authentication

Local authentication controls access to the master key, and therefore to every stored secret. Two gates are available: a four-digit PIN, and platform biometric verification where the device offers it. The active choice is recorded in application preferences as the user authentication type.

PIN is verified by decryption, not by comparison

PIN entry is not checked against a stored hash. The PIN is used to derive the wrapping key and unwrap the master key, and correctness is established by the sentinel and canary checks: a PIN that fails to reproduce a valid master key is by definition wrong. Where the application is resuming and the PIN is still held in memory from the current session, the entered value is compared against that cached copy instead, avoiding a redundant derivation.

Biometric verification

Biometric verification is delegated to the platform prompt. On success the application initialises encryption from the PIN held in the OS keystore, or proceeds directly if the session already holds the key in memory. Incorrect PIN entries are governed by a counter and a timestamp, both held in the OS keystore so that they survive reinstallation of application data. Three thresholds apply:

WRONG ENTRIES	EFFECT
5	A pause of 180 seconds is enforced before each further attempt
7	A blocking warning is displayed, stating attempts remaining
10	All application data is wiped

Session lifetime

Sessions are bounded in time as well as by gating. When the application goes to the background a timestamp is recorded; on return, elapsed time is compared against the effective session lifetime taken from the synchronised user profile, so the policy is administered server-side, with a built-in default applied when no profile has been retrieved. Exceeding it requires re-authentication before the interface becomes available.

Lifecycle cases

Two lifecycle cases complete the picture. On a first installation, enrollment precedes PIN selection, so encryption is initialised with a randomly generated six-digit value that is replaced when the user sets their own PIN. A PIN change verifies the current PIN, re-derives the wrapping key over a fresh salt, and re-wraps the same master key, so stored records are unaffected.

SECURITY PILLAR 02

Data encryption and key management

All stored secrets are encrypted with AES-256 in CBC mode with PKCS#7 padding. A fresh 16-byte initialisation vector is drawn from the platform cryptographic random number generator for every encryption operation, so identical plaintexts never produce identical ciphertexts.

The underlying routine accepts either a password, which it stretches with PBKDF2 over an embedded salt, or a ready AES key; the application uses the second form, passing the master key.

Self-describing ciphertexts

Each ciphertext is self-describing. The stored envelope is a 2-byte format version, a 64-byte random salt, the 16-byte IV, and the ciphertext. Within the encrypted payload the first 128 bytes are a header carrying the format version and a 4-byte magic constant. On decryption the magic and version are checked, and a mismatch is reported as a corrupt-or-wrong-key error. This is the mechanism by which an incorrect PIN is recognised: derivation always succeeds, and it is the sentinel check that fails.

Encryption is applied per field, targeting secret values

RECORD	ENCRYPTED	CLEARTEXT
Server (WebAuthn / FIDO2) credential	credential ID, service name, user handle, signature counter, PKCS#8 private key	record ID, RP ID, display name, timestamps, active flag
Workstation password account	password	login, server URL, domain, type, server account ID
Workstation passwordless account	PIN, HOTP secret	login, server URL, VSC name, counter, use cap, domain
OTP entry	shared secret	account name, issuer, period, digits, hash algorithm

Erasure is cryptographic

Wipe deletes the database, clears preferences and empties the keystore. Because the wrapped master key is destroyed with it, any copy of the database taken beforehand becomes permanently undecryptable; erasure is cryptographic rather than merely a file deletion.

Encryption key management

Master key

At first run the app generates a random 256-bit AES master key. All credential records are encrypted under that key. The key is never stored in the clear: it is wrapped with a 256-bit key derived from the user's 4-digit PIN by PBKDF2 (HMAC-SHA-1) over a 64-byte random salt with 10,000 iterations, and the salt and wrapped key are held together in the OS keystore. Salts and random material are drawn from the platform cryptographic random number generator. Two properties follow: the confidentiality of the database does not depend on PIN entropy, and destroying the wrapped key cryptographically erases all data.

The application manages distinct keys, each with its own generation, storage and lifetime

Master key

AES-256

GENERATED	CSPRNG, on first run
HELD AS	Wrapped, in OS keystore
LIFETIME	Life of the installation

PIN wrapping key

PBKDF2-HMAC-SHA-1 · 10k iterations · 64-byte salt · 256-bit output

GENERATED	Derived on each unlock
HELD AS	Not stored; derived on demand
LIFETIME	Per derivation

Cloud encryption key

RSA-2048, PKCS#1 v1.5

GENERATED	Inside platform keystore, per account
HELD AS	Non-exportable keystore object
LIFETIME	Until account removal

Server (WebAuthn / FIDO2) credential keys

ECDSA P-256

GENERATED	In software, per credential
HELD AS	PKCS#8, encrypted under master key in database
LIFETIME	Until credential deletion

Attestation key

ECDSA secp256r1

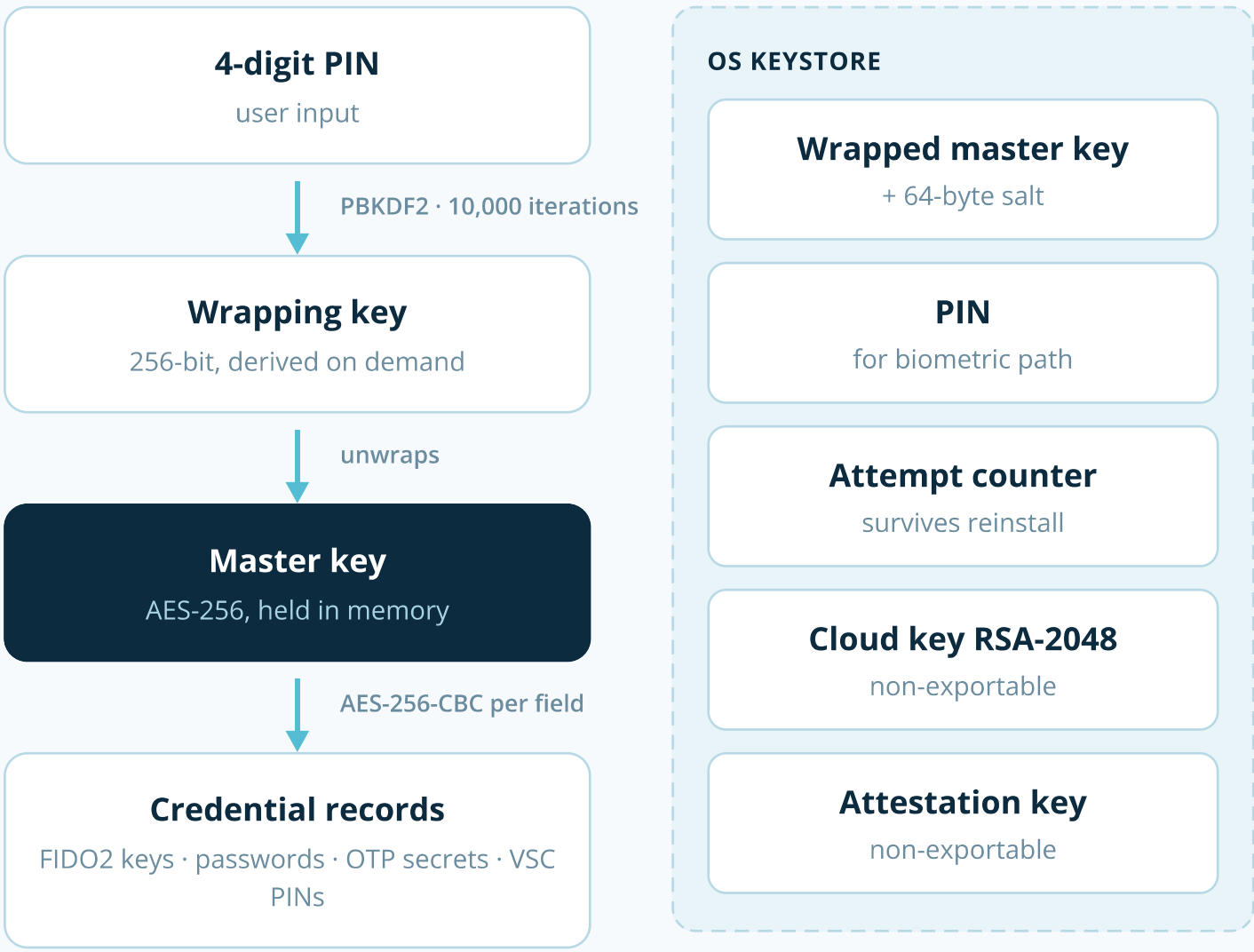
GENERATED	Inside Android Keystore, with attestation challenge
HELD AS	Non-exportable keystore object
LIFETIME	Per enrollment

Key lifecycle

The RSA and attestation keys are generated inside the platform keystore and marked non-exportable, so their private halves are used through keystore operations and never materialise in application memory. The Server (WebAuthn / FIDO2) credential keys are generated in software and stored as PKCS#8, encrypted under the master key.

Changing the PIN re-derives the wrapping key over a fresh salt and re-wraps the same master key, leaving stored records untouched; a new master key is generated only on a fresh installation following a wipe. Key destruction is comprehensive: a wipe empties the keystore, removing the wrapped master key and the PIN, and deletes the keystore objects belonging to each account, while account removal and abandoned enrollment delete their own RSA keys individually.

How the master key is protected



A stolen database without the keystore-held wrapped key is undecryptable; destroying the wrapped key is cryptographic erasure.

SECURITY PILLAR 03

Enrollment process

Enrollment is a WebAuthn registration ceremony that turns an installation into an instance the service recognises. It is driven by a registration request identifier the server issues and delivers out of band. The application uses the WebAuthn credential format and ceremony, scoped to its own service, with keys protected by the application encryption.

The application generates a P-256 key pair on the device and assembles the WebAuthn structures — authenticator data, attestation object, COSE-encoded public key — using fido2-net-lib, the established open-source .NET implementation of WebAuthn/FIDO2, rather than a bespoke implementation of the formats. The AAGUID is a fixed constant identifying the application build, not the device. Evidence of device provenance is supplied separately, over the same registration identifier — on Android a Play Integrity token together with an Android Keystore key-attestation chain conveying verified-boot state and lock status, and on iOS an Apple App Attest object over its SHA-256 hash.

Authenticator data is assembled from the relying-party identifier hash (SHA-256 of the RP ID), a flag byte of credential creation type, a signature counter initialised to zero, and attested credential data carrying the AAGUID, the credential identifier and the COSE-encoded P-256 public key.

A dedicated RSA key for material addressed to one instance

Enrollment also creates a dedicated RSA-2048 key pair inside the platform keystore and transmits only its public half. This is not a sign-in credential: it exists so the service can encrypt material to one specific instance — server tasks and pushed passwords — which the application then decrypts locally. Authentication to the service is by FIDO2 assertion. The private key is non-exportable and never leaves the keystore, so no secret is disclosed during enrollment.

On success the credential is stored with its private key encrypted under the master key. Because enrollment precedes PIN selection on a first installation, encryption is initialised with a temporary generated value and re-wrapped once the user sets a PIN.

Server-side verification

The server, acting as relying party, verifies that the client data type is webauthn.create, that the challenge and origin match those it issued, that the RP ID hash equals the SHA-256 of its own identifier, that the required flag bits are set, and that the credential identifier is not already registered, before storing the public key and signature counter against the account. Device assurance is established from the platform attestation submitted alongside the credential.

NORMATIVE BASIS

WebAuthn / FIDO2 (W3C Recommendation, 8 April 2021): §7.1 registration and the verification steps expected of the server, §6.3.2 authenticatorMakeCredential, §8.7 none attestation. Library: fido2-net-lib.

SECURITY PILLAR 04

Data transfer via the server

All traffic between the application and the service travels over one of two channels, both carried on HTTPS and both authenticated with the JWT: a request/response Web API, and a persistent SignalR hub connection. The hub also carries traffic addressed to Hideez Desktop, which the server relays rather than terminates.

Web API

Request/response calls carry the token as a bearer credential. A small number of calls necessarily precede token issuance — challenge and option retrieval — and the version-compatibility check is deliberately unauthenticated so that an out-of-date client can learn it must upgrade before attempting anything else.

Hub connection

The persistent channel is a SignalR hub connection to the mobileHub endpoint. SignalR is a transport abstraction rather than a wire protocol of its own: client and server negotiate the best transport the network path allows, preferring WebSockets and falling back to Server-Sent Events or long polling where WebSockets are unavailable — behind a restrictive proxy, for example. No transport is excluded by configuration, so the choice is made by negotiation at connection time. Every transport runs over the same HTTPS origin, so transport selection does not alter the protection applied to traffic.

The JWT is supplied through the access-token provider on every connection and reconnection attempt. The server requires a valid token and binds the resulting connection to the account the token identifies. Connection state is tracked and re-established automatically, and each account holds its own connection.

Three purposes of the hub

01

Invoke

It invokes server-side operations directly.

02

Deliver

It delivers server-initiated messages to the application.

03

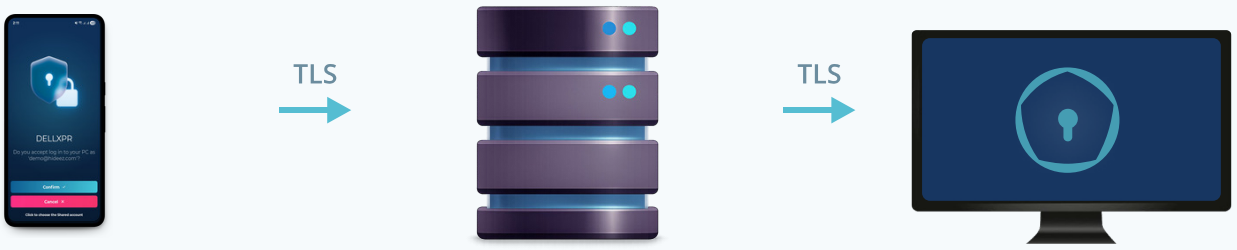
Relay

Most significantly, it relays messages between the phone and Hideez Desktop.

Relay to Hideez Desktop

A relayed message is wrapped in a transfer envelope carrying three fields: the sender's connection identifier, the payload serialised as JSON, and the full type name of the payload. The server forwards the envelope to the recipient identified by the sender. Routing uses an identifier the user obtains by scanning the QR code presented by the credential provider on the workstation, and the destination is determined by the identifier the sender supplies. On receipt, the type name is resolved within the shared-message assembly and the payload deserialised against it, so only message types defined in that assembly can be instantiated.

Each request carries a unique request identifier that the corresponding reply echoes, which is how concurrent operations are correlated. Payloads are protected in transit by TLS on each leg of the path, phone-to-server and server-to-Desktop, with the server performing the relay.



Transfer envelope: sender connection ID · JSON payload · payload type name. Only types defined in the shared-message assembly can be instantiated.

Server tasks

Work queued for the device is pulled over the Web API rather than pushed. Three task types exist: password update, account synchronisation and account deletion. The payloads of the synchronisation and deletion tasks, and the password field of an update task, are RSA-encrypted to the device's cloud encryption key and decrypted inside the platform keystore, so this class of server-originated secret is protected independently of the transport.

Each task is acknowledged to the server only after it has been applied; a task whose execution raises an error is logged and left unacknowledged, so it remains queued rather than being silently consumed. Received passwords are re-encrypted under the master key before storage.

SECURITY PILLAR 05

Bluetooth communication security

The Hideez Authenticator mobile application communicates with Hideez Desktop on a Windows workstation over a dedicated Bluetooth Low Energy service. This communication does not establish an application-layer session and does not apply its own encryption, message authentication, or replay protection.

Bluetooth 5.0 or later is recommended. The basis for the recommendation is implementation maturity rather than additional pairing features: controllers of the 4.2 generation not infrequently implement LE Secure Connections partially despite advertising support, and are far less likely to remain under vendor firmware maintenance — which matters for weaknesses remediated at the controller level rather than in the specification.

Operating-system controls

Beyond the link layer, the protections that apply to this channel are those the operating systems impose on Bluetooth access, and the application is built to sit within them rather than around them. Access to an established connection is mediated per application. A GATT session belongs to the process that opened it, so another application on the same device cannot observe or inject into this application's exchanges with the workstation. This isolation, and not anything in the message format, is what keeps a co-resident application off the channel.

Background operation is deliberately visible. Maintaining a connection while the application is not in the foreground requires a platform foreground service declared for connected-device use, which Android accompanies with a persistent notification, so a live workstation connection is always apparent to the user. The service is not exported, so no other application can bind to it. On iOS the equivalent capability is declared as a background central role and remains subject to the scheduling and lifetime limits the platform applies.

Discovery, requests and authorisation

The workstation advertises a private service identifier together with its own workstation identifier, and the phone matches both before connecting, so it attaches to the specific workstation it intends to operate rather than to any peer offering the service. Exchanges then proceed over a single GATT characteristic, requests being written to it and responses returned as notifications.

A request identifies the operation to perform and carries the data that operation needs; a response indicates success or failure and is correlated with the request that produced it. Authorization is per operation rather than per connection: a request acting on a logon session carries the routing identifier the credential provider issued for that session and presented in the QR code the user scanned, and Hideez Desktop validates it before acting. This binds a request to one logon session, which limits what a captured request can be replayed against; it is an authorisation check and not a cryptographic replay-protection mechanism.

SECURITY PILLAR 06

Workstation login and unlock with Hideez Desktop

An unlock is always initiated at the workstation and completed from the phone, using passwordless or password-based Windows accounts.

Hideez Desktop presents a QR code containing the routing identifier for the pending logon session together with the workstation identity and, for server-mediated flows, its connection identifier. The user scans it, selects which credential to use, and the application delivers that credential to the workstation over whichever transport is available.

Transports

Two exist, and they differ only in carriage, not in what the workstation receives.
Over Bluetooth the phone connects directly to the workstation and writes an unlock request, as described in Bluetooth Communication Security. Over the server the credential travels as a relayed message through the hub, as described in Data Transfer via the Server. The relayed path requires both devices to reach the service; the Bluetooth path requires only radio proximity, and is the only one that functions when the workstation is offline.

Password-based unlock

The stored password is decrypted on demand under the master key, combined with the login in its legacy form and the phone's name, and sent with the routing identifier. The workstation validates the routing identifier, performs the logon and returns a result; a missing or unsuccessful response is surfaced as a distinct error rather than a silent failure. The plaintext password exists only for the duration of the operation.

Passwordless unlock

The virtual smart card's PIN is decrypted the same way and sent with the user name, domain, the phone's identifier and the routing identifier. Because the card and its certificate reside on the workstation, what crosses the link is the PIN that unlocks the card rather than a reusable account password.

Offline unlock

Where neither transport is available, the credential can still be used through a code the user transcribes. The application derives an offline code on the HOTP construction from the secret shared with the workstation during setup and displays it; the workstation validates it against its own copy without contacting either the phone or the service. Codes are single-use, and the supply available from a given secret is bounded.

SECURITY PILLAR 07

QR code based authentication on HES

The credential established at enrollment serves two purposes, both of which are WebAuthn authentication ceremonies over a server-supplied challenge: authorising a sign-in request raised elsewhere — typically a workstation or web login the user approves by scanning its QR code — and obtaining the bearer token the application itself uses for API calls. The two differ only in which endpoint issues the challenge and what the server does with the result.

Constructing an assertion

To construct an assertion the application decrypts the credential identifier, the PKCS#8 private key, the signature counter and the user handle from the database under the master key. It increments the counter, then assembles authenticator data from the relying-party identifier hash (SHA-256 of the RP ID), the flag byte, the incremented counter and the credential data. Client data is serialised as JSON containing the type webauthn.get, the base64url-encoded server challenge, the origin, and crossOrigin: false. The signature is computed with ECDSA over the P-256 curve and SHA-256, over the concatenation of the authenticator data and the SHA-256 hash of the client data, and is encoded as a DER sequence. Authenticator data, signature, client data and user handle are submitted together with the credential identifier and the device operating system, model and application version.

Monotonic signature counter

The signature counter is monotonic and advances by one on every assertion. The incremented value is persisted on the device, re-encrypted under the master key, and the server likewise advances and records the counter for the credential on each authentication, so a counter that fails to advance as expected is detectable server-side.

Token lifecycle

A successful token exchange returns a JWT, which is stored in the OS keystore under a key derived from the credential identifier, so each enrolled account holds its own token. The application reads the token’s expiry and arms a single-shot timer at half the remaining lifetime; when it elapses, or when a token is found absent or expired, a fresh challenge is fetched and a new assertion performed to mint a replacement. Role information is read from the token’s claims. Revocation discards the in-memory token, cancels the timer and removes the stored copy, so the next call must re-assert.

NORMATIVE BASIS

WebAuthn / FIDO2 (W3C Recommendation, 8 April 2021): §7.2 verification of an authentication assertion, §6.3.3 authenticatorGetAssertion, §6.1.1 signature counter considerations.

SECURITY PILLAR 08

Root check and jailbreak detection

Device integrity is addressed at two points: a local check each time the application starts, and platform attestation collected at enrollment and evaluated by the service.

Android — launch-time root check

Release builds test for indicators of a rooted system before the interface is constructed. Three independent probes are applied and any one is treated as positive.

1 Build tags
Inspects the build tags for test-keys, which distinguishes a development or custom system image from a release-signed one.

2 Known su locations
Tests ten known locations for the su binary and the Superuser package: /sbin, /system/bin, /system/xbin, /system/sd/xbin, /system/bin/failsafe, /data/local and its bin and xbin subdirectories, /su/bin, and /system/app.

3 Path resolution
Executes which su and treats any output as an indication that the binary is resolvable on the path.

If any probe reports positively the application does not start: it displays a notice that rooted devices are not supported and terminates its activity stack, so no credential store is opened and no interface becomes reachable.

Android — enrollment attestation

Two further signals are gathered during enrollment, both bound to the registration identifier created during device enrollment so that neither can be replayed from an earlier session. The first is a Play Integrity token, requested through the standard integrity API against the application’s cloud project, with the SHA-256 hash of the registration identifier supplied as the request hash. The second is an X.509 certificate chain produced by generating a fresh elliptic-curve key on the secp256r1 curve inside the Android Keystore with the registration identifier set as its attestation challenge; the resulting chain is signed by the platform and carries the root-of-trust information for the device, including its verified-boot state and whether the bootloader is locked. Both are transmitted with the enrollment request and evaluated by the service, which is where the decision on device acceptability is made.

iOS

No equivalent filesystem probe is performed. Device integrity on iOS is established through Apple's App Attest service: where the platform provides it, the application generates an attestation key and attests it against the SHA-256 hash of the same registration identifier, and the resulting attestation object is submitted with the enrollment request for evaluation by the service.

EVERY START

Launch-time check

Governs whether the application runs at all on a given device, and is applied on every start.

AT ENROLLMENT

Platform attestation

Establishes, at the moment an installation is bound to an account, what the platform itself reports about the integrity of the device and the authenticity of the application build.

The two layers answer different questions. The launch-time check is applied on every start; the attestation signals are evaluated once, when an installation is bound to an account.

What each platform supplies at enrollment

Android

Play Integrity token

Requested through the standard integrity API against the application's cloud project, with the SHA-256 hash of the registration identifier as the request hash.

Keystore key attestation

A fresh secp256r1 key generated inside the Android Keystore with the registration identifier as its attestation challenge; the resulting X.509 chain is signed by the platform.

Conveys

Verified-boot state · whether the bootloader is locked.

iOS

Apple App Attest

The application generates an attestation key and attests it against the SHA-256 hash of the same registration identifier.

No filesystem probe

Device integrity is not established locally; it rests on the platform attestation service.

Conveys

Authenticity of the application build on a genuine device.

HES

Both signals travel with the enrollment request. The service evaluates them — the decision on device acceptability is made server-side, never on the device.

Attack vectors and mitigations

Phishing and social engineering

Adversary-in-the-middle

THREAT

A proxy page relays a one-time code or an approval to the genuine service in real time to hijack the session.

MITIGATION

Sign-in to the Hideez server involves no code the user types or reads out: the phone signs a server-issued challenge, and the resulting assertion is single-use and tied to a request whose options expire after five minutes, so nothing transcribable exists for a proxy to capture. Approval is correlated to the browser session by that request identifier alone, so the flow does not itself distinguish the user's own session from a relayed one. One-time-password codes for third-party services are transcribed by the user and remain relayable, as their protocols determine that.

Device and application compromise

Local malware and infostealers

THREAT

Malicious software attempts to read stored secrets or session material.

MITIGATION

Secrets are encrypted under the master key; the wrapped key and the PIN are held in the OS keystore, and the RSA and attestation private keys are non-exportable within it. No application component is exported, and inbound deep links carry only server-issued request identifiers. Abuse of screen-reading interfaces requires an explicit, user-visible platform grant.

Physical theft or lock-screen bypass

THREAT

An attacker in possession of the device uses the application directly.

MITIGATION

Access does not depend on the device lock screen: the application applies its own PIN or biometric gate, and that gate releases the master key. Attempts are counted in the keystore, a pause follows five failures and the store is erased after ten. Erasure destroys the wrapped master key and is therefore cryptographic. Sessions expire while backgrounded.

Protocol and system weaknesses

Flawed implementation

THREAT

Defects in cryptographic handling or authorisation logic permit unauthorised local access to the vault.

MITIGATION

Standard constructions and platform facilities are preferred to bespoke ones: WebAuthn structures come from an established open-source implementation, keys are generated by the platform keystore or platform primitives, and all record encryption passes through a single component. Messages from the server deserialise only to types defined in the shared-message assembly. A canary record confirms the derived key before any credential is touched.

Compromised operating system

THREAT

The device is rooted, which is the precondition for extracting keystore material.

MITIGATION

Release builds test for indicators of a rooted system at every start and terminate without opening the credential store.

Interception between the application and the server

THREAT

An attacker on the network path reads or alters traffic, including messages relayed to the workstation.

MITIGATION

Both legs are HTTPS; the hub connection is authenticated with the application's token and bound by the server to the account it identifies; Web API requests carry the same token. Secrets the server pushes are additionally encrypted to the device's non-exportable RSA key, so that material is protected independently of the transport.

ABOUT HIDEEZ

Hideez provides innovative IAM solutions ensuring both secure and usable authentication

Hideez Authenticator gives every employee a corporate credential without issuing them anything — passwordless MFA for web accounts and passwordless Windows login for the shared workstations your teams rotate through.

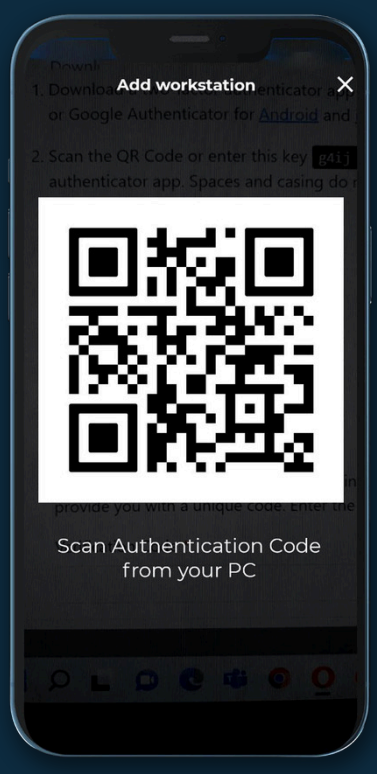
There is no hardware to buy, ship or replace: staff use the phone they already carry. You approve each workstation, hand out and revoke login methods from one console, and every sign-in on a shared account still carries a named person in the audit trail.

Employees can focus on their work, while Hideez takes care of their authentication and compliance with security protocols. Hideez Authentication Solution integrates with multiple websites and apps, closing the gap of passwordless SSO adoption.

Let’s start a discussion together

Find out more about us: www.hideez.com

BOOK A DEMO



© 2026 Hideez Group. All rights reserved. · www.hideez.com