inlockly

SECURITY MODEL AND CRYPTOGRAPHY

Inlockly™ Security Whitepaper

How Inlockly protects your data: the zero-knowledge model, on-device encryption, account processes step by step, access recovery, and the limits of protection.

Version 2.1 · October 5, 2026

© 2026 Inlockly™ · All rights reserved

Contents

01 Key points at a glance 3
02 Introduction 5
03 Fundamental data protection principles 6
04 What the server sees - and what it doesn't 8
05 Cryptographic technologies 11
06 Infrastructure and transport security 16
07 Master Password 17
08 Creating an account 19
09 Signing in 25
10 Two-factor authentication (2FA) 29
11 Vaults 31
12 Changing the Master Password 35
13 Changing the email address 39
14 Recovery key 43
15 Account reset without a recovery key 48
16 Restoring access to groups after a reset 52
17 Comparing access recovery paths 55
18 Data sharing 56
19 Helper features and privacy 58
20 Threat model and the limits of protection 60
21 Recommendations 62
22 Glossary 63
23 Standards and sources 65
24 Changelog 66

© 2026 Inlockly™ · inlockly.com

01

Key points at a glance

Inlockly™ is a password manager built on a **zero-knowledge** model. Passwords, notes, 2FA codes, and other data in your vault are encrypted on your device before they reach the server. The server only stores and syncs ciphertext - it doesn't know your Master Password or any key that could decrypt it.

This document explains how that works: which algorithms we use, how keys are created and protected, what happens when you sign up, sign in, unlock your vault, or change your password and email address, how access recovery and group sharing work - and what no password manager can protect against.

Property	How we do it
Data encryption	AES-256-GCM on your device, with a new random vector for every write
Key from password	Argon2id (64 MiB, 4 iterations, 4 threads) - every attempt to guess a password is expensive
Authentication	The server only receives a one-way hash, which it hashes again with BCRYPT
Sharing	Group vault keys are shared using RSA-OAEP encryption (2048-bit, SHA-256)
Access recovery	A recovery key known only to you; without it - an account reset with a 7-day waiting period
Second factor	Email code, authenticator app, YubiKey hardware key, and backup codes
Transport	Every connection to the server is encrypted with TLS

Property	How we do it
Data location	Servers in the European Union

New in version 2.1

We now describe the **recovery key** (recovering your account after forgetting your Master Password without losing data), **account reset without a key** with admin-restored access to groups, and extended two-factor authentication, plus new chapters: “What the server sees,” the threat model, recommendations for administrators, and a glossary. The full list of changes is in the “Changelog” chapter.

02

Introduction

Cybersecurity experts recommend using unique, randomly generated passwords for every account you create online. Long, complex passwords are secure, but extremely hard to remember. That's why many people use weak, easy-to-remember passwords. A study by the UK's National Cyber Security Centre (NCSC) found that millions of people sign in to online services with passwords containing predictable phrases, such as a pet's name or a favorite band, as well as "password" and "123456" (see the "Standards and sources" chapter for details).

Passwords like these leave accounts exposed to attacks: cybercriminals crack weak passwords in seconds using automated tools. An 11-character password made up only of digits can be cracked almost instantly. Add lowercase and uppercase letters and special characters, and the time needed to crack it grows to decades.

Uniqueness leaves a lot to be desired, too. People often reuse one password for many accounts. Cracking that password gives a criminal access to a large amount of sensitive data, and a leak puts at risk not only you but also your company, family, or team. On top of that, workplaces often share passwords in insecure ways - in messages, spreadsheets, or on sticky notes.

Inlockly™ helps solve these problems. It provides tools for creating, storing, and sharing passwords and sensitive information for individuals and groups, while maintaining the highest level of security. Data syncs automatically and securely across every device the user has authorized.

2.1 Scope and audience

This whitepaper is intended for users, IT administrators, security teams, and auditors who want to understand how Inlockly™ protects data. It describes the application as of the release date of this version of the document. We present detailed processes in diagrams - each one comes with a step-by-step description.

Throughout the document, we capitalize the proper names of keys (for example, Master Key, Symmetric Key). You'll find their definitions in the "Glossary" chapter.

03

Fundamental data protection principles

The Inlockly™ application relies on key security measures to protect all user data. These include:

3.1 End-to-end encryption

End-to-end encryption blocks access to your passwords, data, and private information through a comprehensive implementation of AES-256-GCM encryption, key derivation with Argon2id, and salted hashing. Your data is readable only on your devices.

3.2 Local-only encryption model

All cryptographic keys are generated and managed solely by the application on the user's device. The same goes for every encryption and decryption operation - they happen locally, on an authorized device. The server doesn't perform any cryptographic operation on vault data.

3.3 Zero-knowledge encryption

Members of the Inlockly™ team can't see your passwords or private data. The key that opens your vault is derived from your Master Password and email address - on your device.

Your Master Password is never stored on or sent to Inlockly™ servers. The same goes for the recovery key you receive when you sign up.

3.4 Privacy by design

Inlockly™ stores your logins and private data as an encrypted vault, synced across all your devices. Before data leaves your device, it's fully encrypted, so only you can access it. Our developers and other Inlockly™ staff have no way to read your data - even if we wanted to. Features that could reveal the contents of your vault run locally: for example, checking passwords against known breaches happens on your device, without sending passwords or their hashes to outside services.

3.5 Secure data sharing

Inlockly™ lets you create groups (family, team, company) in which users share data securely. A combination of symmetric and asymmetric encryption protects data during sharing: each member gets their own encrypted copy of the group vault key, which only they can read. The server stores these copies but can't open them.

3.6 Security audits

We regularly review and security-test the Inlockly™ application. We run routine checks of our services and our entire infrastructure. The application complies with data protection laws, including the GDPR.

3.7 Rapid incident response

The Inlockly™ team responds to every reported incident. We investigate, analyze, and fix the bugs or potential vulnerabilities identified in a report. We inform users of every significant incident on our website and in the release notes for the new version of the app.

3.8 Bug bounty program

Inlockly™ runs a bug bounty program in which security researchers can report vulnerabilities they find. We offer rewards for finding and documenting vulnerabilities. We follow clear rules and ask researchers to practice responsible disclosure - reporting the bug to the Inlockly™ team and **not disclosing it publicly** until we've confirmed it has been fixed.

04

What the server sees - and what it doesn't

A zero-knowledge model doesn't mean the server knows nothing. To sync data, manage plans, and enforce group permissions, the service needs to know some administrative information. Below, we honestly separate one from the other.

4.1 Data encrypted on your device

This data reaches the server only as ciphertext. The server has no key to read it:

Data	Encrypted with
Vault item contents: name, username, password, addresses, notes, custom fields, 2FA keys	Symmetric Key (private vault) or group key
Item metadata: favorites, trash, keys for active sharing links	as above
Item change history (previous versions)	as above
The account's Symmetric Key	Master Key (from the Master Password) and - separately - a key from the recovery key
RSA Private Key	Symmetric Key
Copies of group vault keys	The member's Public Key (RSA-OAEP)
Autofill settings for websites	Symmetric Key
Content shared via link	A one-time link key that never reaches the server

4.2 Data the server knows

Data	Why
First name, last name, and email address	Account, sign-in, security notifications
BCRYPT hash of the Master Password Hash	Authentication without knowing the password
Plan, subscription, billing	Plan entitlements (payments are handled by a payment processor)
Names of groups, departments, and folders; memberships and permissions	Access control in groups
Vault structure: item IDs, folder, author and time of change, version number	Sync and change history
Devices and sessions: OS, app, IP address, approximate location, time	Device limits, new sign-in alerts, activity log
Account and group activity log	Transparency for users and administrators
2FA method settings	Verifying the second factor (see the 2FA chapter)

Why aren't folder and group names encrypted?

The server enforces group permissions: it decides who can see which folder and sends notifications about changes. So we encrypt the content, not the organizational structure. Don't put sensitive information in the names of folders, departments, or groups - keep it in vault items.

4.3 What the server never receives

- Your Master Password - not at sign-up, not at sign-in, and not when you change your password.
- The Master Key, Symmetric Key, Private Key, or group keys in plain form.
- Your recovery key - the server only knows a proof derived from it, stored under an HMAC seal.
- A sharing link's key - it's in the part of the address after the `#` sign, which browsers never send.
- Passwords being checked for breaches - the comparison happens on your device.

05

Cryptographic technologies

Inlockly™ uses only proven, standardized algorithms available in the platforms' native cryptographic libraries (WebCrypto). Argon2id, which WebCrypto doesn't offer, is implemented by the reference library compiled to WebAssembly.

5.1 Parameters at a glance

Element	Algorithm and parameters
Master Key	Argon2id: 64 MiB memory, 4 iterations, 4 threads, 256-bit output; salt: the first 16 bytes of SHA-256 of the normalized email address
Master Password Hash	PBKDF2-SHA256, 1 iteration, payload: Master Key, salt: Master Password; 256-bit output
Password hash on the server	BCRYPT, cost 12
Symmetric Key	256 bits from a CSPRNG
Data and key encryption	AES-256-GCM, 96-bit nonce from a CSPRNG (new for every encryption), 128-bit tag
Ciphertext format	`ILK1.` + nonce + ciphertext with tag (base64url)
Key pair	RSA-OAEP, 2048-bit, SHA-256 (OAEP and MGF1)
Group key	256 bits from a CSPRNG
Recovery key	128 bits from a CSPRNG, written as 28 Crockford base32 characters with a 12-bit checksum

Element	Algorithm and parameters
Keys from the recovery key	HKDF-SHA256, empty salt, contexts “inlockly recovery v1 wrap” and “… auth”
Local unlock	HMAC-SHA256 (key: Master Password Hash, message: email address)
Vault backup (.inlockly)	Argon2id with a random salt and parameters stored in the file, AES-256-GCM
Transport	TLS

5.2 Key hierarchy

Keys form a chain - each one is protected by the one before it. That way, changing your Master Password or email address only requires re-encrypting one short value, not your entire vault.

1. **Master Password** (only in the user’s memory) → Argon2id produces the **Master Key**.
2. The **Master Key** encrypts the account’s **Symmetric Key** (the Protected Symmetric Key).
3. The **Symmetric Key** encrypts the items in the private vault and the RSA **Private Key**.
4. The **Private Key** decrypts the copies of **group keys**, and those encrypt the items in group vaults.
5. Independently of step 2, the **recovery key** encrypts a second copy of the same Symmetric Key.

5.3 AES encryption standard

The Advanced Encryption Standard (AES) is a modern symmetric block cipher - one of the most widely used in the world. It was published by Vincent Rijmen and Joan Daemen, and in 2001 it was adopted as a U.S. federal standard (FIPS PUB 197). It’s also described in ISO/IEC 18033-3.

Inlockly™ uses the AES-256 variant, where 256 is the key length in bits. Encryption with a 256-bit key consists of 14 rounds of transformations applied to each 128-bit block of data. The cipher is symmetric: the same key is used to encrypt and decrypt the data.

GCM mode

AES-256 runs in GCM (Galois/Counter Mode), defined in NIST SP 800-38D. GCM provides authenticated encryption: a single operation ensures both the confidentiality and the integrity of the data. A 128-bit authentication tag is attached to every ciphertext. Any change to the encrypted data, even a single bit, makes decryption fail, and the modified data is rejected.

Every encryption operation uses a new random 96-bit initialization vector (nonce) from a CSPRNG - even when the same item is saved again. Identical data produces a different ciphertext every time, so the encrypted form doesn't reveal whether two items have the same content.

5.4 RSA algorithm

The Rivest-Shamir-Adleman (RSA) algorithm is one of the most widely used asymmetric cryptography algorithms. It creates two related keys: a public key that anyone can use to encrypt a message, and a private key that only its owner can use to read it. RSA's security rests on the difficulty of factoring large composite numbers.

Inlockly™ uses RSA with a 2048-bit key and the OAEP padding scheme (Optimal Asymmetric Encryption Padding) with the SHA-256 hash function, as defined in PKCS#1 v2.2 (RFC 8017). OAEP adds randomness to every encryption and protects against known attacks on the older PKCS#1 v1.5 padding.

In Inlockly™, RSA doesn't encrypt the data itself. It's used only to securely hand group members the random 256-bit group vault key, which AES uses to encrypt the data. This combination, called hybrid encryption, pairs the advantages of asymmetric cryptography (no need to exchange a secret key) with the performance and strength of a symmetric cipher.

5.5 Argon2id function

Argon2 is a password-based key derivation function that won the international Password Hashing Competition in 2015. It's standardized in RFC 9106. The Argon2id variant combines resistance to side-channel attacks with resistance to attacks using graphics cards and

specialized hardware (ASICs). It's the variant recommended by RFC 9106 and OWASP.

Unlike functions such as PBKDF2, whose cost depends only on the number of iterations, Argon2id requires a large amount of memory for every computation. An attacker trying to guess a password can't cheaply run thousands of attempts in parallel: every attempt costs as much memory as a single user sign-in.

The parameters (64 MiB of memory, 4 iterations, 4 threads) match the configuration RFC 9106 recommends for memory-constrained devices, plus one extra iteration. They're constants in the application and aren't stored on the server.

> **Why 64 MiB and not more?**
>
> The Master Password is also needed in system extensions - for example, in iPhone autofill, which has a hard memory limit per process. A higher memory cost would mean the system shuts down the extension mid-unlock. The chosen parameters meet or beat the industry norm: one iteration more than the recommended configuration.

5.6 PBKDF2, HKDF, and HMAC

PBKDF2-SHA256 is used only to derive the Master Password Hash from the finished Master Key, in a single iteration (see the "Creating an account" chapter). **HKDF-SHA256** (RFC 5869) derives two independent keys from the random recovery key. **HMAC-SHA256** lets the app verify the Master Password during local vault unlock, without involving the server.

5.7 BCRYPT algorithm

BCRYPT is a password hashing algorithm with a built-in salt and a key-stretching mechanism. The Inlockly™ server uses it to hash the received Master Password Hash again (cost 12) - so even a database leak doesn't reveal a value that could be sent at sign-in.

5.8 Random number generation - CSPRNG

A Cryptographically Secure Pseudo-Random Number Generator (CSPRNG) provides unpredictable values: keys, initialization vectors, and the recovery key. Inlockly™ uses the operating system's generator exposed by the platform.

5.9 TLS protocol

TLS (Transport Layer Security), the successor to SSL, secures transmission between the app and the server (RFC 8446). In Inlockly™, it's a second, independent layer of protection: data sent over TLS is already encrypted on the device.

06

Infrastructure and transport security

The Inlockly™ service runs on servers located in the European Union. The apps connect to the server only over encrypted TLS connections.

The security of vault data doesn't depend on infrastructure: the server stores ciphertext it has no keys for. Taking over the server, a backup, or the database doesn't give access to vault contents. Infrastructure is, however, responsible for service availability, the integrity of administrative data, and protection against abuse. That's why the server uses, among other things:

- **attempt limits** for sign-in (6 per minute from one IP address and 20 per minute for one email address), recovery, and account reset - each path has its own independent counter;
- **bot protection** (Cloudflare Turnstile) at sign-up, sign-in, and when starting recovery;
- **an HMAC seal** on everything that determines access (session tokens, trusted device tokens, 2FA settings, backup codes) - writing to the database without knowing the server's secret doesn't let anyone bypass the password or the second factor;
- **responses resistant to account enumeration** - sign-up, sign-in, recovery, and reset return the same response whether or not the address has an Inlockly™ account.

Infrastructure details

We'll publish a detailed description of the infrastructure architecture (availability, backups, data processing regions) in the next version of this whitepaper.

07

Master Password

When creating an Inlockly™ account, the user sets their **Master Password**. It's used for authentication, and above all, it's the source - on the device - of the keys that open the user's private vault and the vaults of the groups they belong to.

The vault is where logins, passwords, 2FA codes, and other sensitive data are stored. That's why the Master Password should be strong, and the account should also be protected with two-factor authentication (2FA).

7.1 A strong Master Password

When you create an account, the app shows a password strength meter and won't accept a password rated as weak. Your Master Password should be long and unique. A good choice is a phrase made of several random words, or a longer string of different characters: lowercase and uppercase letters, numbers, and special characters.

The strength of your Master Password directly affects the security of your data. Argon2id makes every attempt to guess the password expensive, but it's no substitute for a good password: a short or predictable password can still be guessed despite the high cost of each attempt.

> ⚠ Important
>
> **Never use your Master Password on another website or app.** A data breach at another service could then put your vault at risk.
>
> **Never share your Master Password or recovery key** - including with people claiming to be Inlockly™ staff. Nobody on our team needs to know them, and we'll never ask you for them. Treat any such request - by email, phone, or chat - as a phishing attempt and report it to us.

7.2 Protecting the Master Password

Encrypted vault data is useless without the key derived from the user's Master Password and email address. The Master Password is wiped from the app's memory right after the keys are derived and **is never sent** to the Inlockly™ server. The server only receives a one-way hash (the Master Password Hash), from which neither the password nor the keys can be reconstructed.

As a result, nobody on the Inlockly™ team can access your Master Password or read the encrypted data in your vault or in the vaults of the groups you belong to.

7.3 If you forget your Master Password

Because your Master Password never reaches the server, we can't remind you of it or reset it like a typical online service would. Since version 2.1, however, you have two ways out - both consistent with the zero-knowledge model:

- **Recovery key** - you receive it when you sign up. It lets you set a new Master Password and keep all your data: your private vault, your keys, and your access to groups. Described in the "Recovery key" chapter.
- **Account reset without a key** - when you have neither your password nor your key. It keeps your account, plan, and group memberships, but your private vault is deleted, because nobody can decrypt it. Access to group data is restored by the group's administrator. Described in the "Account reset" and "Restoring access to groups" chapters.

08

Creating an account

Sign-up is when all of an account's keys are created. It happens entirely on the user's device - the server only receives hashes and ciphertext.

In short:

- the **Master Key** (Argon2id) and the **Master Password Hash** (PBKDF2) are derived from the Master Password and email address;
- a random **Symmetric Key** encrypts the private vault, and is itself encrypted with the Master Key;
- an **RSA key pair** lets the user securely receive group vault keys;
- the **recovery key** protects a second copy of the Symmetric Key in case the password is forgotten;
- the server hashes the Master Password Hash with BCRYPT and stores the keys unchanged - it can't read them.

8.1 Account creation process diagram

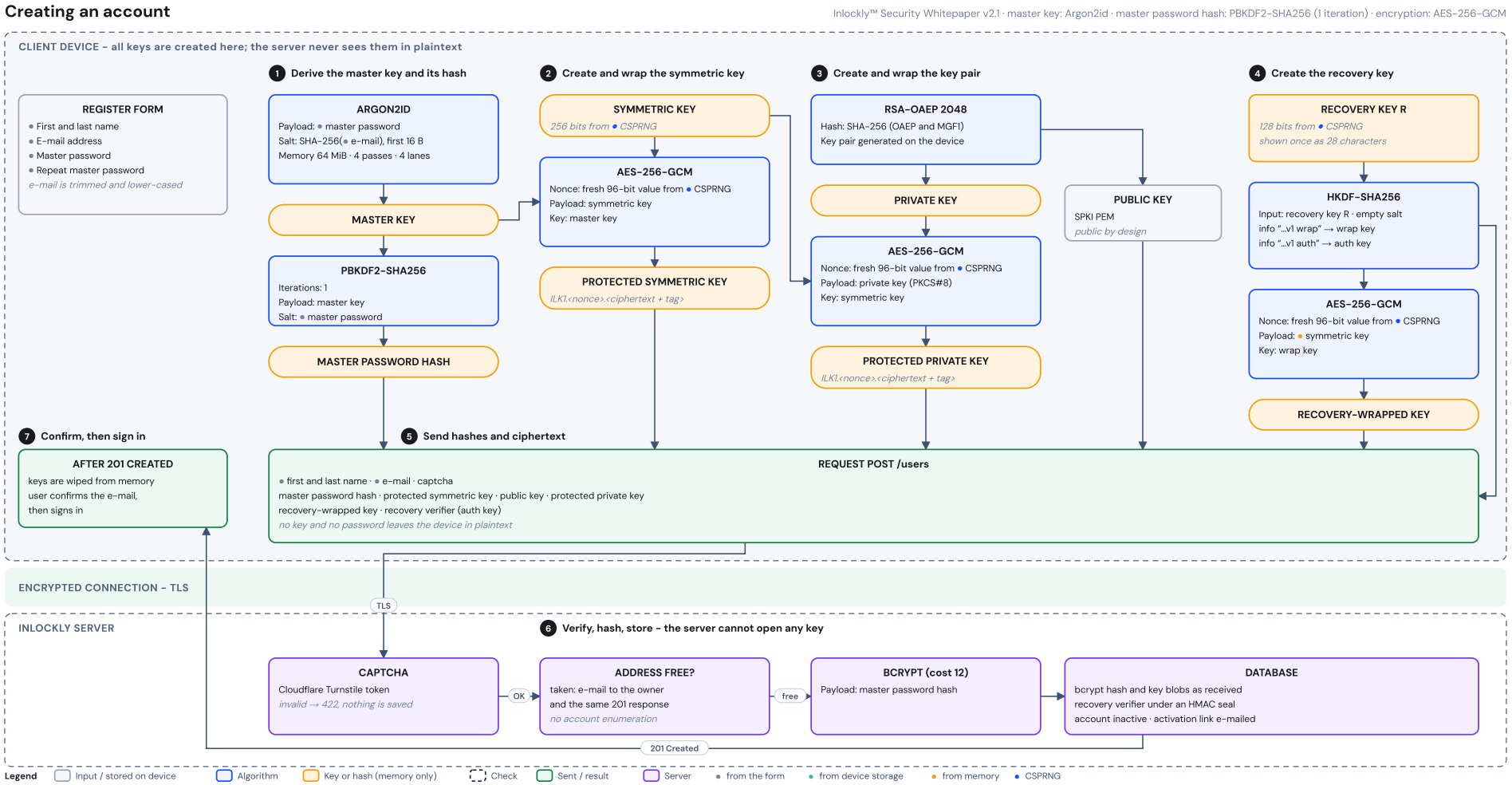


Figure 1. Creating an account: all keys, including the recovery key, are created on the user's device.

8.2 Account creation, step by step

Sign-up form

First, the user fills in the sign-up form, entering:

- first and last name,
- email address,
- Master Password,
- Master Password again.

The user accepts the terms of service and can opt in to marketing communications. The form is protected against bots (Cloudflare Turnstile). The Master Password is entered twice to rule out typos - the app compares both fields on the device. The strength meter shows how resistant the password is to guessing, and the app won't accept a password rated as weak.

The email address is normalized: the app trims extra spaces and converts letters to lowercase. The same address therefore always produces the same keys, no matter how it's typed. All the cryptographic operations described below are performed on the user's device before any data is sent to the server.

Deriving the Master Key

The Master Key is derived locally from the Master Password using Argon2id. In a single operation, the function mixes the password with a salt and stretches it so that every attempt to guess the password is expensive. The salt is the SHA-256 hash of the normalized email address (the first 16 bytes). The other parameters: 64 MiB of memory, 4 iterations, and 4 threads. The result is a 256-bit Master Key, used to encrypt the Symmetric Key.

The Master Key is never stored on or sent to Inlockly™ servers.

Generating the Symmetric Key

The Symmetric Key is a random 256-bit value from a CSPRNG. It encrypts the data in the private vault and the Private Key. To store it securely, the app encrypts it with AES-256-GCM using these parameters:

- initialization vector (nonce): a new random 96-bit value from a CSPRNG,

- payload: Symmetric Key,
- key: Master Key.

The result is the Protected Symmetric Key, which is sent to the server. Reading it requires the Master Key, and only someone who knows the Master Password can derive that.

Why isn't data encrypted directly with the Master Key?

The Master Key comes from the Master Password and the email address, so it changes whenever either of them changes. If data were encrypted directly with the Master Key, every password or address change would require decrypting and re-encrypting the entire vault - a long operation that could be interrupted. That's why data is encrypted with a random Symmetric Key that stays the same for the lifetime of the account. When the password or address changes, only that one short value is re-encrypted.

Why is the Protected Symmetric Key sent to the server?

The Symmetric Key is needed on every one of the user's devices. Its encrypted form is stored on the server and handed over after sign-in on a new device. Decryption happens only on the device and requires the Master Key - knowing the ciphertext alone doesn't let anyone read the data.

Generating the RSA key pair

The app generates an RSA Key Pair on the device (2048-bit, OAEP with SHA-256). It's used to securely pass along the group vault keys that let users share data with each other.

The **Public Key** is public by design. When User A adds User B to a group, they encrypt the group vault key with B's Public Key and give B only the encrypted copy. Only B's Private Key can decrypt it. The server stores this copy but can't read it.

The **Private Key** decrypts the copies of group keys, and with them the data in every vault shared with the user. That's why it's protected just like the Symmetric Key: it never leaves the device in plain form. Before sending it to the server, the app encrypts it with AES-256-GCM using a new initialization vector, and the key is the **Symmetric Key**. The result is the Protected Private Key. Because it's protected by the Symmetric Key rather than the Master Key, changing the password or email address doesn't require re-encrypting the Private Key.

Master Password Hash

The Master Password Hash is used for authentication - the server doesn't need the Master Password for that and never receives it. We derive it with PBKDF2-SHA256 using these parameters:

- iterations: 1,
- payload: Master Key,
- salt: Master Password.

The result is a 256-bit Master Password Hash. A single iteration is a deliberate choice: Argon2id already protects against password guessing, and the Master Key the hash is derived from is a 256-bit value that can't be guessed. The hash only needs to be one-way - a second Argon2id call would make sign-in slower with no security benefit.

Creating the recovery key

Before the account is created, the app generates a **recovery key** - 128 random bits from a CSPRNG. The user sees it as 28 characters in seven groups of four and has to keep it outside Inlockly™ (by copying, downloading, or printing it). The account isn't created until the user confirms they've saved the key.

HKDF-SHA256 derives two independent keys from the recovery key: the **wrap key** encrypts (AES-256-GCM) a second copy of the Symmetric Key, and the **auth key** becomes a verifier that lets the server check that the user knows the recovery key without learning the key itself. Details are in the "Recovery key" chapter.

Sending the data and the server's role

Over an encrypted TLS connection, the app sends the server the user's first and last name, email address, and bot-protection token, plus the cryptographic material: the Master Password Hash, the Protected Symmetric Key, the Public Key, the Protected Private Key, the copy of the Symmetric Key for recovery, and the recovery key verifier. No key or password leaves the device in plain form - except the Public Key, which is public by design.

The server first checks the bot protection, then whether the email address is already taken. If it is, the server doesn't create an account, emails the owner of the address about the sign-up attempt, and returns the same response as for a successful sign-up - so it's impossible to tell whether a given address has an Inlockly™ account. If the address is free, the server hashes the Master Password Hash with BCRYPT (cost 12), stores the verifier under an HMAC seal, and stores the keys without any modification, because it can't read them. The account stays inactive until the email address is confirmed with an activation link.

After receiving the response, the app wipes all keys from memory and stores nothing on the device. The user confirms their email address and then signs in as described in the “Signing in” chapter.

09

Signing in

Signing in happens in two steps: the server checks whether the user knows the Master Password (by comparing hashes), and the device - which has the Master Key - decrypts the account keys itself. Passing verification on the server alone isn't enough to read any data.

In short:

- only the Master Password Hash reaches the server - never the password or the Master Key;
- an account with 2FA turned on requires a second factor, unless the device is trusted;
- the server sends back the account keys only in encrypted form;
- the device stores an HMAC code that later lets it unlock the vault without contacting the server.

9.1 Sign-in process diagram

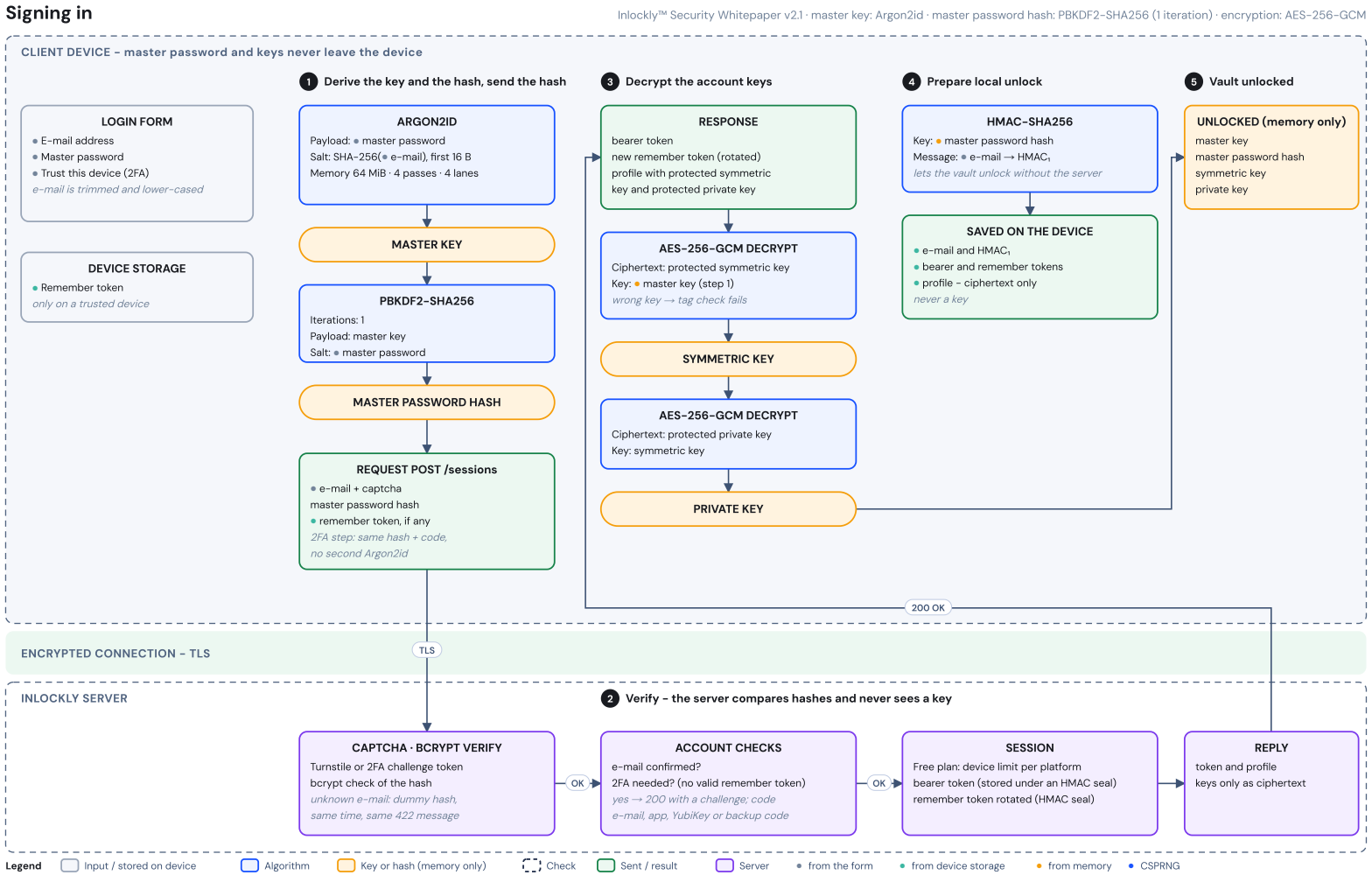


Figure 2. Signing in: the server compares hashes and sends back keys only in encrypted form.

9.2 Sign-in, step by step

The user fills in the sign-in form with their email address and Master Password. The form is protected against bots (Cloudflare Turnstile), and the email address is normalized the same way as at sign-up.

1. **Deriving the keys and sending the Master Password Hash.** The app derives the Master Key from the Master Password with Argon2id (salt: the SHA-256 hash of the email address, same parameters as at sign-up). The Master Key stays only in memory. The Master Password Hash is derived from the Master Key and the Master Password (PBKDF2-SHA256, one iteration). Over TLS, the app sends the email address, the Master Password Hash, and the bot-protection token. Neither the Master Password nor the Master Key reaches the server.

2. **Verification on the server.** The server checks the bot protection and compares the received Master Password Hash with the stored BCRYPT hash. If the address doesn't exist or the hash doesn't match, it returns the same invalid credentials message. For an unknown address, it compares against a dummy hash, so the response time is the same - there's no way to tell whether the address has an account. Attempts are rate-limited: 6 per minute from one IP address and 20 per minute for one email address.

   Next, the server checks whether the account's address has been confirmed and whether the account has two-factor authentication turned on while the device isn't marked as trusted. If so, instead of a session it returns a challenge for the second factor (see the "Two-factor authentication" chapter). The app then sends the same Master Password Hash again along with the code - without recomputing Argon2id.

   Finally, the server creates a device session, taking the plan's device limit into account. It issues a session token, plus a trust token on a trusted device; it stores both in the database only under an HMAC seal. A successful sign-in cancels any pending account reset (see the "Account reset" chapter). In its response, it sends the tokens and account data, including the Protected Symmetric Key and the Protected Private Key - the keys only in encrypted form.

3. **Decrypting the account keys.** The app decrypts the Protected Symmetric Key with AES-256-GCM using the Master Key from step 1, then uses the Symmetric Key to decrypt the Protected Private Key. The GCM authentication tag guarantees that a wrong key or a modified ciphertext results in an error, not a misread value. Authenticating with the server alone isn't enough to read the data - you need to know the Master Password.

4. **Preparing local unlock.** The app computes an HMAC-SHA256 code where the key is the Master Password Hash and the message is the email address. It stores the code, email address, tokens, and account data (encrypted only) on the device. This later lets the vault be unlocked without the server (see the “Vaults” chapter). No key is stored on the device.

5. **Vault unlocked.** The Master Key, Master Password Hash, Symmetric Key, and Private Key stay only in memory. The app downloads the encrypted items and decrypts them on the device: private vault items with the Symmetric Key, and group vault items with the group keys previously decrypted with the Private Key.

Sign-in notifications

Signing in on a new device triggers an email with details of the attempt: operating system, IP address, country, and time. If it wasn’t you, change your Master Password and check the device list in your account settings.

10

Two-factor authentication (2FA)

Two-factor authentication (2FA) verifies the user with two different forms of identification: something they know (the Master Password) and something they have (an email inbox, a phone, a hardware key). Knowing the address and password is then not enough to sign in to your account.

10.1 Second-factor methods

Method	How it works	Availability
Email code	A one-time 6-digit code sent to the account address, valid for 15 minutes	Every plan
Authenticator app (TOTP)	A 6-digit code that changes every 30 seconds (RFC 6238), e.g. from Google Authenticator	Personal plan and up
YubiKey hardware key (OTP)	A one-time password generated by a physical key when you touch it	Personal plan and up
Backup codes	10 one-time codes in case you lose your phone or key	Personal plan and up

You can add several methods to an account - at sign-in, the user picks whichever is convenient. Methods set up on a paid plan stay active even after moving to the Free plan: downgrading never weakens your account's security. On the Free plan, you just can't add a new method or generate new codes.

10.2 Second-factor safeguards

- **Attempt limit:** 5 incorrect app codes or backup codes lock the second factor for 15 minutes. The counter is shared, so backup codes aren't a second, independent way to guess. The 2FA step is only reached after a correct password, so a lockout triggers an

email recommending a Master Password change.

- **Replay protection:** the same app code won't be accepted twice.
- **Backup codes** use the format `xxxx-xxxx`, are single-use, and are stored under an HMAC seal. Using a code triggers an email with the number left; when three or fewer remain, the app recommends generating a new set. A new set fully invalidates the previous one.
- **Trusted device:** after you check "Trust this device," the server issues a trust token that skips the 2FA step on that device. The token rotates at every sign-in, and account recovery and account reset invalidate all trusted devices.

> **Is the authenticator app secret covered by zero-knowledge?**
>
> No - and that's a deliberate difference. To verify the code at sign-in, the server has to know the account's TOTP secret. It's stored encrypted with the server application's key, outside the database. This applies only to the second factor for signing in to Inlockly™. **The 2FA codes you save in vault items** for other services are part of the encrypted item content and are calculated only on your device.

10.3 Losing your second factor

> **⚠ Important**
>
> Keep your backup codes outside Inlockly™, so they're available even when you can't sign in. If you use hardware keys, we recommend adding at least two to your account: one you carry with you and a spare.

If you lose access to your authenticator app or key, sign in with a backup code and then set the method up again. Inlockly™ support doesn't turn off 2FA manually - that kind of workaround would be a back door for anyone impersonating the account owner. The access recovery paths (recovery key and account reset) also require the second factor if the account has one.

11

Vaults

A vault is a set of data encrypted with the same key. After creating an account, every user becomes the owner of a **private vault**. **Group vaults** are created within groups of users (family, team, company) who share data with each other.

11.1 Encrypting data in vaults

Items in the **private vault** are encrypted with the user's Symmetric Key (AES-256-GCM). This key is created when the account is set up and is protected by the Master Key derived from the Master Password.

A **group vault** has its own random 256-bit key, generated when the group is created. Each member gets their own copy of this key, encrypted with their Public Key (RSA-OAEP). Only that member has the Private Key that decrypts their copy, and with it, the data in the group vault.

The entire item content is encrypted (name, username, password, addresses, notes, custom fields, 2FA keys), along with its metadata and change history. Every write uses a new initialization vector.

11.2 Locking vaults

The vault locks manually, after a period of inactivity (5 minutes by default), when the screen locks, or when the system goes to sleep. Locking doesn't sign you out of your account, but it wipes all cryptographic keys from memory. To use the app again, just enter your Master Password (or, on a phone, use biometrics).

You can change the inactivity timeout in your preferences. Instead of locking, you can also choose to sign out - then using the app again requires a full sign-in, plus the second factor if it's turned on.

11.3 Vault unlock process diagram

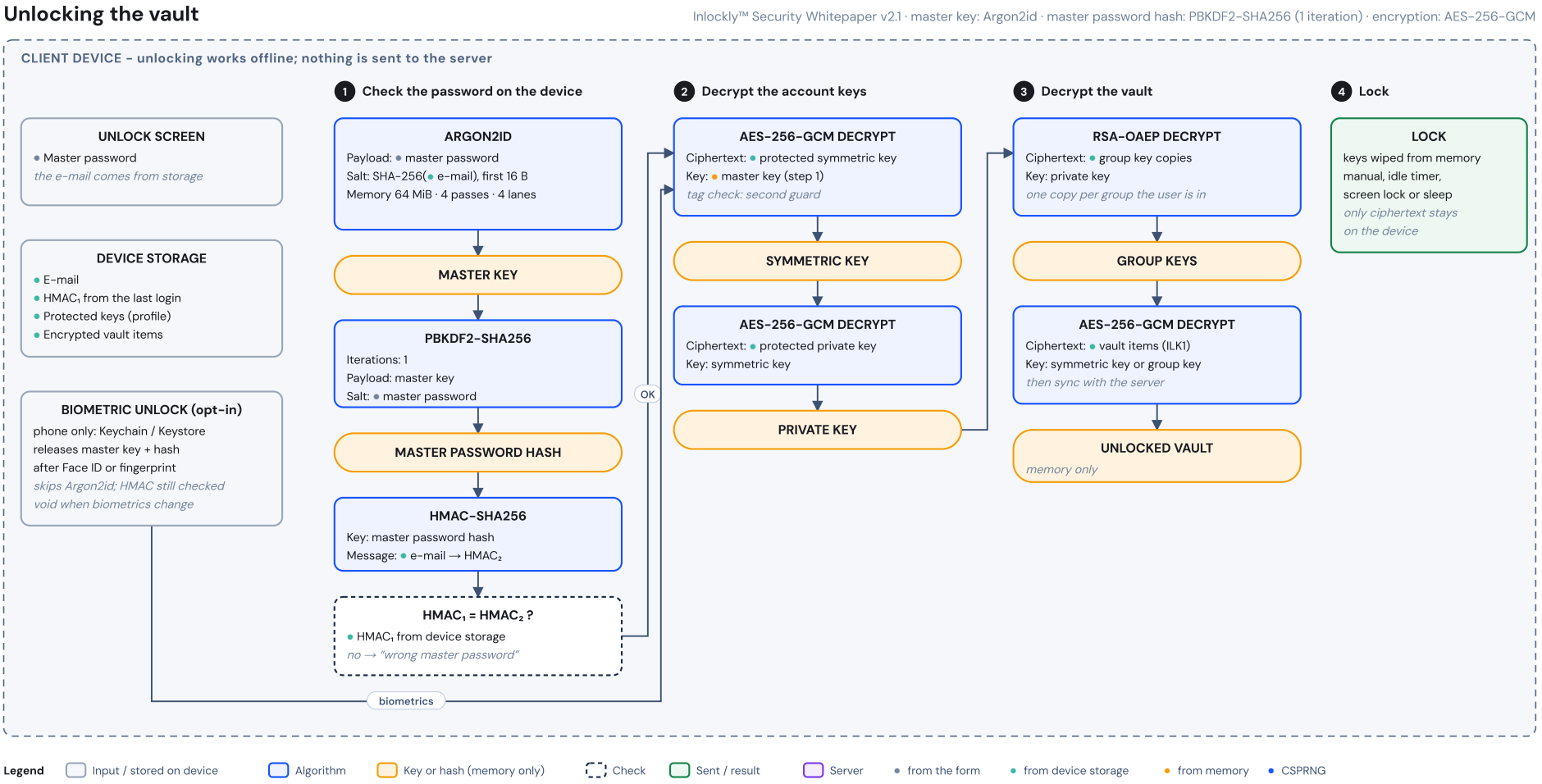


Figure 3. Unlocking the vault happens entirely on the device – it works even without an internet connection.

11.4 Vault unlock, step by step

A locked app shows a form where the user enters their Master Password. No email address is needed - the user hasn't been signed out, so the address is in the device's storage (partially masked on screen). Unlocking happens entirely on the device, without the server.

1. **Checking the password on the device.** The Master Key is derived from the Master Password (Argon2id, salt from the email address stored on the device), and the Master Password Hash from that. The app computes an HMAC-SHA256 code and compares it with the code stored at the last sign-in. If the codes don't match, it reports an incorrect Master Password and doesn't try to decrypt the keys. The stored code doesn't make guessing the password any easier, because every attempt requires a full Argon2id computation.
2. **Decrypting the account keys.** The app decrypts the Protected Symmetric Key with the Master Key, then the Protected Private Key with the Symmetric Key. The GCM authentication tag provides a second, independent check: a wrong key or a modified ciphertext makes decryption fail.
3. **Decrypting the vaults.** With the Private Key, the app decrypts (RSA-OAEP) the copies of the group keys, then the items stored locally: private vault items with the Symmetric Key, and group vault items with the matching group key. When there's an internet connection, the app syncs the vaults with the server, sending and receiving only encrypted data.
4. **Locking.** The vault stays unlocked until it's locked manually, the inactivity timeout runs out, the screen locks, or the system goes to sleep. Locking wipes all keys from memory.

11.5 Biometric unlock

On a phone, the user can turn on unlocking with Face ID, Touch ID, or a fingerprint (Personal plan and up). Once turned on, the Master Key and the Master Password Hash are stored in the system's hardware-protected key store (Keychain on iOS, Keystore on Android) with these safeguards:

- the system releases them only after a successful biometric check, every time they're used;
- they're never included in backups and can't be moved to another device;
- they're permanently invalidated when the set of enrolled fingerprints or faces on the system changes.

Biometric unlock skips the Argon2id computation but still checks the HMAC code. After the Master Password or email address is changed on another device, the stored secret becomes outdated - the app detects this from the code mismatch, turns off biometrics, and asks for the Master Password. The secret is deleted on sign-out, when the session expires, and when the password or address changes. This is the only case in which the Master Key leaves the app's memory - and only at the user's explicit request.

12

Changing the Master Password

Changing the Master Password is fast and secure no matter how big your vault is: only the Symmetric Key

- one short value – is re-encrypted. The data in your vaults, the Private Key, and the recovery key stay the same.

The current password is checked twice: first on the device (without contacting the server), then on the server. After the change, your other devices are signed out and the account owner gets an email notification.

12.1 Master Password change process diagram

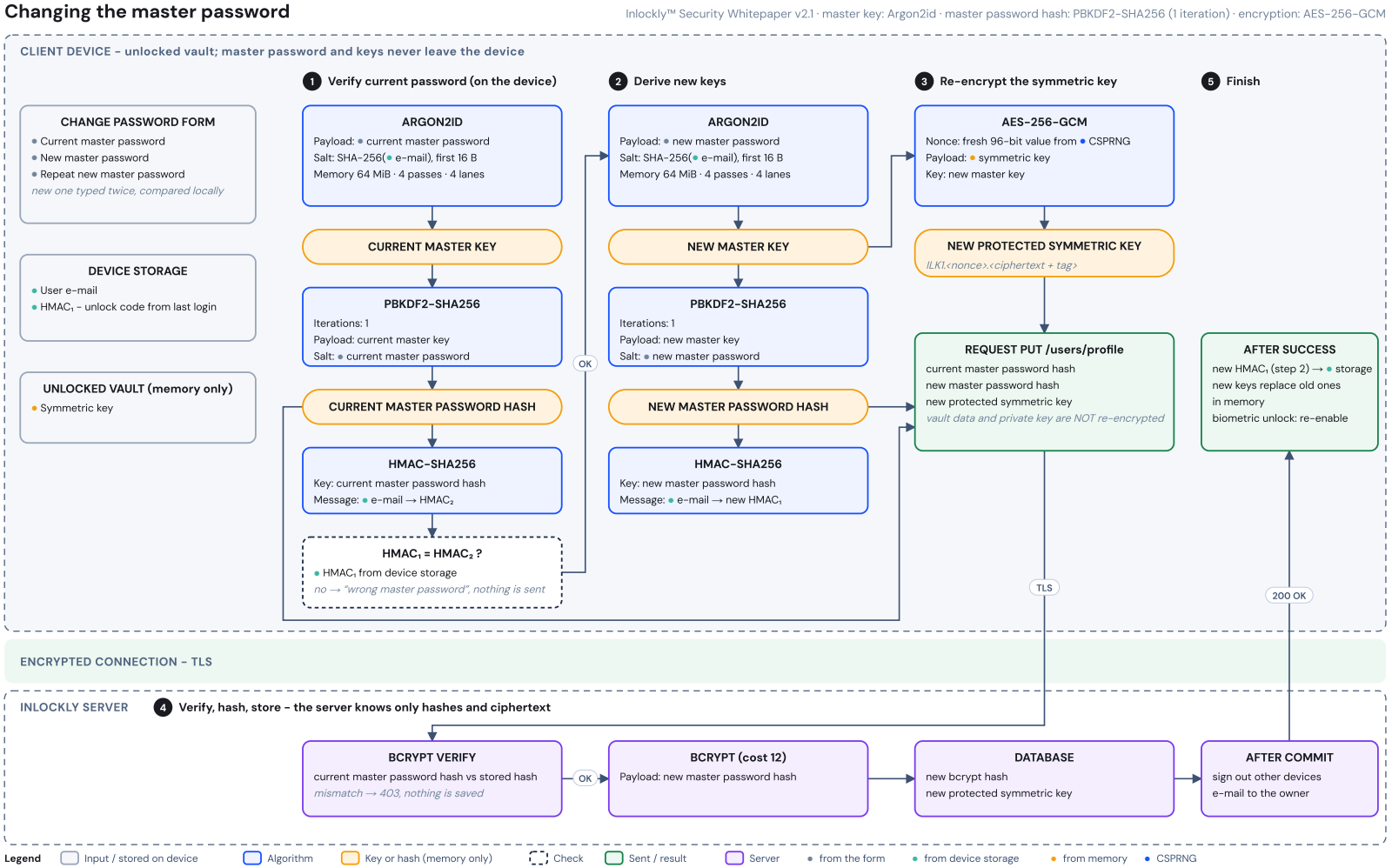


Figure 4. Changing the Master Password: only the Symmetric Key is re-encrypted - vault data stays the same.

12.2 Changing the Master Password, step by step

In account settings, the user fills in the password change form with their current Master Password and the new Master Password twice. The new password is entered twice to rule out typos - the app compares both fields on the device. The current password confirms that the account owner is making the change: first on the device, then on the server. The change happens while the vault is unlocked, so the Symmetric Key is already in memory and isn't derived again.

1. **Verifying the current password on the device.** The Current Master Key is derived from the current Master Password (Argon2id, salt from the email address stored on the device), and the Current Master Password Hash from that. The app computes an HMAC-SHA256 code and compares it with the code stored at the last sign-in. If the codes don't match, it reports an incorrect password, and no request is sent to the server.

2. **Deriving the new keys.** In the same way, the New Master Key is derived from the new Master Password (same salt, because the email address doesn't change), along with the New Master Password Hash and a new HMAC code.

3. **Re-encrypting the Symmetric Key.** The Symmetric Key is encrypted with AES-256-GCM using the New Master Key and a new initialization vector. The result is the New Protected Symmetric Key - the only element that needs re-encrypting. The Private Key and vault data are encrypted with the Symmetric Key, which doesn't change.

4. **Verification and storage on the server.** Over TLS, the app sends the Current Master Password Hash, the New Master Password Hash, and the New Protected Symmetric Key. The server re-authorizes the operation by comparing the Current Hash with the stored BCRYPT hash; if they don't match, it rejects the request and stores nothing. After successful verification, it hashes the New Hash with BCRYPT (cost 12) and stores it along with the New Protected Symmetric Key. It then signs out the other devices and sends the owner an email notification. Throughout the process, the server works only with hashes and ciphertext.

5. **Finishing up.** The app stores the new HMAC code on the device and replaces the keys in memory with the new ones. If the user was using biometric unlock, it's turned off (the stored secret was based on the previous Master Key) - it can be turned on again in settings.

Your recovery key stays valid

Changing the Master Password doesn't affect the recovery key: it protects the same Symmetric Key, which doesn't change. If you suspect someone has also learned your recovery key, generate a new one in your account settings - the previous one will stop working.

Does changing the password undo the effects of a leak?

Changing your password cuts off access for anyone who only knows the old password. It isn't, however, a replacement of the key your data is encrypted with. If someone previously had access to your unlocked vault, they could have read its contents - in that case, besides changing your Master Password, also change the passwords stored in your vault.

13

Changing the email address

The email address is the salt for Argon2id, so changing it means a new Master Key and a new Master Password Hash – even though the Master Password stays the same. The new address has to be confirmed with a code, and a notification about the change goes to both the old and the new address.

13.1 Email address change process diagram

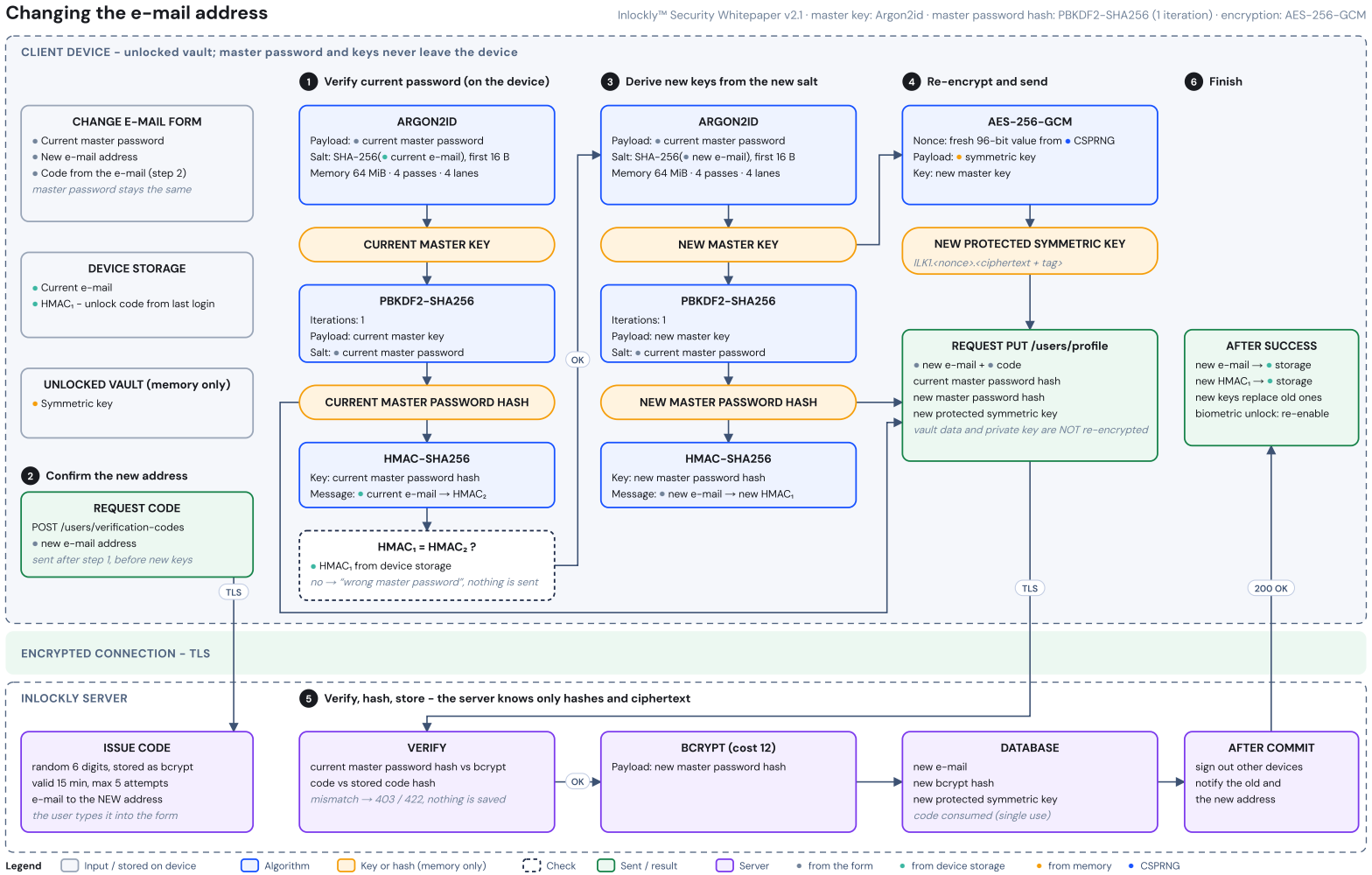


Figure 5. Changing the email address: a new address is a new Argon2id salt, so new keys are created.

13.2 Changing the email address, step by step

In account settings, the user enters their current Master Password, the new email address, and the verification code sent to the new address. The current password confirms that the account owner is making the change. The change happens while the vault is unlocked, so the Symmetric Key is already in memory.

Changing the email address is also a change of keys: the Argon2id salt is a hash of the email address, so the same Master Password combined with a new address gives a different Master Key.

1. **Verifying the current password on the device.** The Current Master Key is derived from the Master Password (salt from the current address), and the Current Master Password Hash from that. The app compares the HMAC-SHA256 code with the code stored at the last sign-in. If the codes don't match, no request is sent to the server.

2. **Confirming the new address.** The app normalizes the new address and asks the server for a verification code. The server generates a one-time 6-digit code tied to the target address, stores only its protected form, and sends the code to the new address. The code is valid for 15 minutes and is invalidated after 5 incorrect attempts. By entering it, the user confirms access to the new inbox.

3. **Deriving new keys from the new salt.** The New Master Key is derived from the same Master Password - this time the salt is a hash of the new address - and from it the New Master Password Hash and a new HMAC code.

4. **Re-encrypting the Symmetric Key and sending the data.** The Symmetric Key is encrypted (AES-256-GCM) with the New Master Key and a new initialization vector. Over TLS, the app sends in a single request: the new address, the verification code, the Current and New Master Password Hashes, and the New Protected Symmetric Key.

5. **Verification and storage on the server.** The server compares the Current Hash with the BCRYPT hash and checks the code. If any condition fails, it rejects the request. After successful verification, it stores the new address, the new password hash, and the New Protected Symmetric Key, consumes the code, signs out the other devices, and sends a notification to both the old and the new address - so the owner finds out about the change even if they didn't make it themselves.

6. **Finishing up.** The app stores the new address and the new HMAC code, replaces the keys in memory with the new ones, and turns off biometric unlock. At the next sign-in, the user enters the new address and their existing Master Password.

Your recovery key stays valid

The recovery key protects the Symmetric Key, which doesn't change - it keeps working after you change your email address.

14

Recovery key

The recovery key lets you recover your account after forgetting your Master Password - **without losing any data**. It keeps your private vault, your Private Key, and your access to all your groups. Every user gets one when they sign up, on every plan.

The key doesn't restore your old password. It opens a second, independent copy of the Symmetric Key, so the user can set a new Master Password. The server doesn't know the recovery key - the zero-knowledge model remains intact.

14.1 What the key looks like

- **128 random bits** from a CSPRNG, generated on the device - the user doesn't choose them.
- Format: **28 Crockford base32 characters**, in seven groups of four, with a **12-bit checksum** that catches typos when the key is retyped (the checksum doesn't add to the key's strength).
- When entering it, case doesn't matter, and confusable characters are normalized (O as 0, I and L as 1).

> **Why HKDF and not Argon2id?**
>
> Argon2id slows down guessing of passwords that people make up. The recovery key is 128 random bits - it can't be guessed, so HKDF-SHA256 (RFC 5869) is enough to derive keys with separate purposes from it.

14.2 Two keys from one secret

From recovery key R, HKDF-SHA256 (empty salt) derives two 256-bit values with different contexts:

- the **wrap key** (context "inlockly recovery v1 wrap") - encrypts, with AES-256-GCM, the copy of the Symmetric Key that is sent to the server;

- the **auth key** (context “inlockly recovery v1 auth”) - proof of knowing the recovery key. The server stores it as a verifier under an HMAC seal and never returns it in any response.

Separating the contexts means the proof sent to the server can’t be used to decrypt the copy of the account key.

14.3 Account recovery process diagram

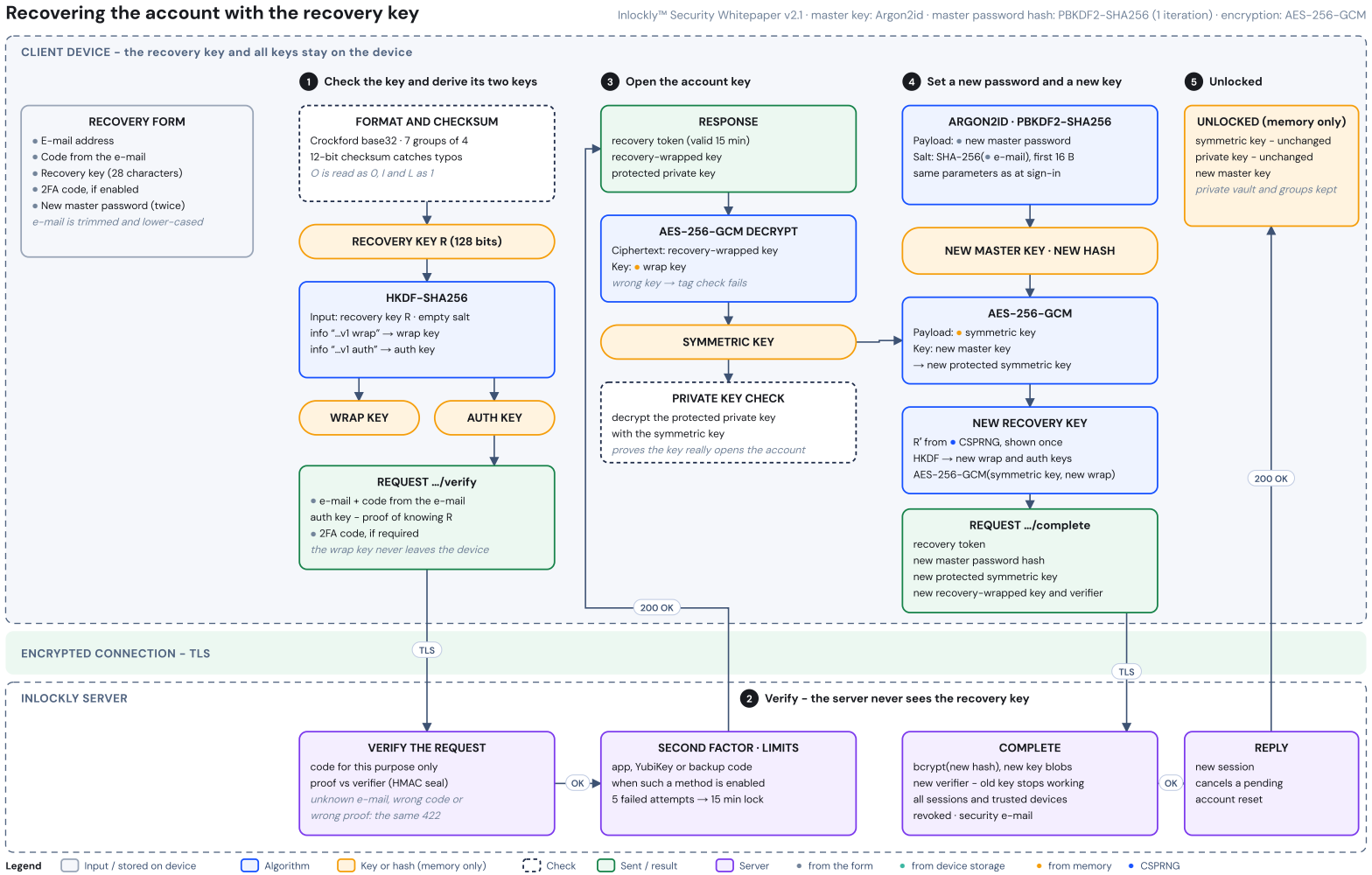


Figure 6. Recovering an account with the recovery key: your vault, keys, and groups are preserved.

14.4 Account recovery, step by step

On the sign-in screen, the user chooses “Forgot your master password?” and enters their email address (the form is protected against bots). The server always responds the same way and sends a verification code only if the account exists and has a recovery key - so this can’t be used to check who has an account.

1. **Checking the key and deriving two keys.** The app checks the key’s format and checksum, then derives the wrap key and the auth key. Over TLS, it sends the address, the code from the email (meant only for recovery), the auth key as proof, and - if the account has an authenticator app or a YubiKey - the second-factor code (or a backup code). The wrap key never leaves the device.

2. **Verification.** The server checks the code and compares the proof with the verifier. An unknown address, a wrong code, and a wrong proof all produce the same response. Five incorrect attempts lock recovery for that address for 15 minutes. After successful verification, the server consumes the code and returns a short-lived recovery token (valid for 15 minutes), the copy of the Symmetric Key for recovery, and the Protected Private Key - keys in encrypted form only.

3. **Opening the account key.** The app decrypts the copy of the Symmetric Key with the wrap key (AES-256-GCM) and checks that the Symmetric Key really does open this account’s Private Key.

4. **A new password and a new recovery key.** The New Master Key and the New Master Password Hash are derived from the new Master Password - just as when changing the password. The Symmetric Key is encrypted with the New Master Key. The app also generates a **new recovery key** (shown once) along with a new copy and a new verifier. Everything is sent to the server together with the recovery token.

5. **Storage and a new session.** In a single transaction, the server stores the new password hash and the new ciphertext, replaces the verifier (the old recovery key stops working), invalidates all sessions and trusted devices, cancels any pending account reset, and sends an email about the account recovery with details of the attempt. The app gets a new session and unlocks the vault right away.

© 2026 Inlockly™ · inlockly.com

> ⚠ Keep your recovery key outside Inlockly™
>
> A key saved only in your vault won't help when you forget your password - the vault will be locked. Print it or save it on a storage device kept somewhere safe. Never send it to Inlockly™ staff, an administrator, or in a support ticket.

14.5 Replacing the key and its limits

- You can generate a new key in your account settings (requires your Master Password) - the previous one stops working completely, and the owner gets a security email.
- Every use of the key ends with a new one being issued - the old one works once.
- The recovery key **doesn't replace the second factor or access to your email inbox** - recovery requires both if the account has 2FA turned on. That's why you should store your backup codes so they don't depend on a locked account.
- Replacing the key removes its path in the service, but doesn't undo an earlier exposure: if someone had both the recovery key and previously copied ciphertext, they could have read it outside the service.

15

Account reset without a recovery key

Account reset is the path for someone who doesn't know their Master Password **and** doesn't have their recovery key. It isn't data recovery: the private vault can't be decrypted, because nobody - Inlockly™ included - has the key to it. What a reset does is let you keep your account, email address, plan and subscription, and group memberships by creating an entirely new set of keys. Access to group data is then restored by the group's administrator (see the "Restoring access to groups" chapter).

A reset is irreversible for the private vault, so it's protected by a **seven-day waiting period** with daily notifications. An email inbox and a second factor can be lost together - for example, with a stolen phone - and the waiting period gives the owner time to react.

15.1 Account reset process diagram

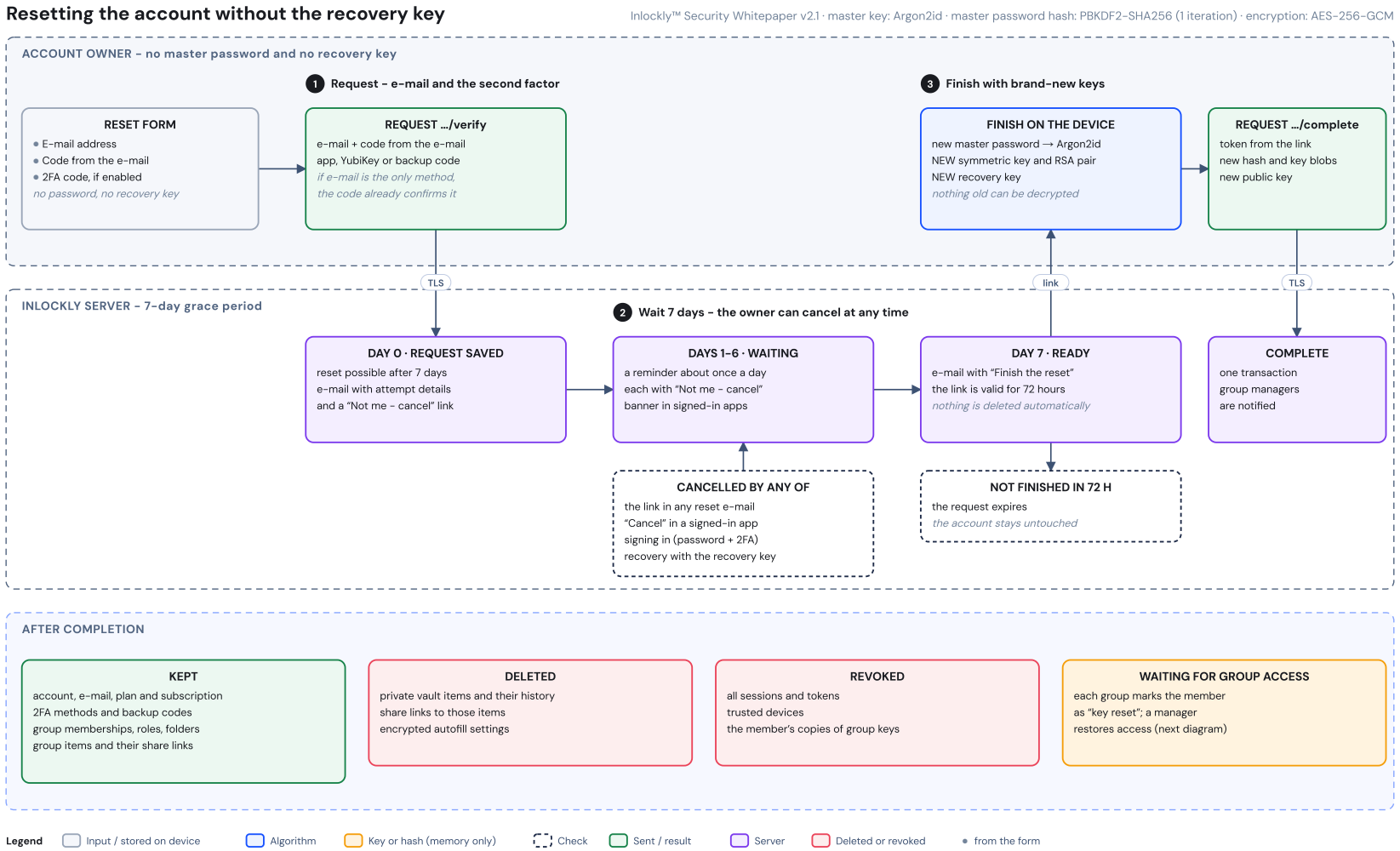


Figure 7. Account reset: a request, a 7-day waiting period you can cancel, and completion with new keys.

15.2 Account reset, step by step

1. **Request.** The user enters their email address (the form is protected against bots) and the code sent to that address. If the account has an authenticator app or a YubiKey, they also enter the second-factor code or a backup code; when email is the only 2FA method, the code from this step already confirms it. An unknown address, a wrong code, and a wrong second factor all produce the same response, and incorrect attempts lock the path just like recovery does. Support staff can't submit a request on the user's behalf.

2. **Waiting period - 7 days.** The server records the request and immediately sends an email with details of the attempt and a "This wasn't me - cancel the reset" link. Over the following days, it sends a reminder roughly once a day, and signed-in apps show a banner with a cancel button. Any of the following cancels the request:

    - the link in any reset email,
    - "Cancel" in a signed-in app,
    - a successful sign-in (password and second factor) - anyone who knows the password doesn't need a reset,
    - recovering the account with the recovery key.

3. **Completion on the device.** After 7 days, the user gets an email with a "Complete reset" link, valid for 72 hours. **Nothing happens automatically** - if the link isn't used, the request expires and the account stays untouched. After clicking, the app creates every key from scratch: a new Master Password and Master Key, a new Symmetric Key, a new RSA key pair, and a new recovery key (shown once). Everything is created on the device, just like at sign-up.

4. **Completion on the server.** In a single transaction, the server stores the new password hash and the new keys, deletes data encrypted with the old keys, invalidates sessions, and marks group memberships as waiting for restored access. It sends an email confirming the reset, and the managers of each group get a message asking them to confirm the user's identity before restoring access.

15.3 What a reset changes

Area	Result
Account, email address, plan, subscription	Kept

Area	Result
2FA methods and backup codes	Kept - a reset isn't a way to get around 2FA
Group memberships, roles, departments, folders, group ownership	Kept
Group items and the links sharing them	Untouched (encrypted with group keys)
Private vault items and their change history	Deleted
Sharing links for private items, autofill settings	Deleted
Sessions, tokens, and trusted devices	Invalidated
This person's copies of group keys	Deleted - waiting to be restored by a manager

Sync forces a full download of the current state on every device, because local copies are encrypted with the old keys. Deleting data from the service's database doesn't mean it's immediately removed from infrastructure backups - those follow a separate retention policy, and they only contain ciphertext anyway.

Vault backup

If you previously made an encrypted backup of your vault (an `.inlockly` file with a separate password), you can import it after the reset. It's an independent path - a reset neither creates nor deletes it.

16

Restoring access to groups after a reset

After a reset, the user still belongs to their groups, with the same permissions, departments, and folders. However, their new Private Key doesn't open the copy of the group key encrypted for their previous key pair – that copy was deleted. Access is restored by someone else who has their own working copy of the group key.

The server plays no cryptographic role in this: it doesn't decrypt anything or generate a replacement key. The administrator also doesn't gain access to the user's private vault – it was lost in the reset, and nobody can rebuild it.

© 2026 Inlockly™ · inlockly.com

16.1 Access restoration process diagram

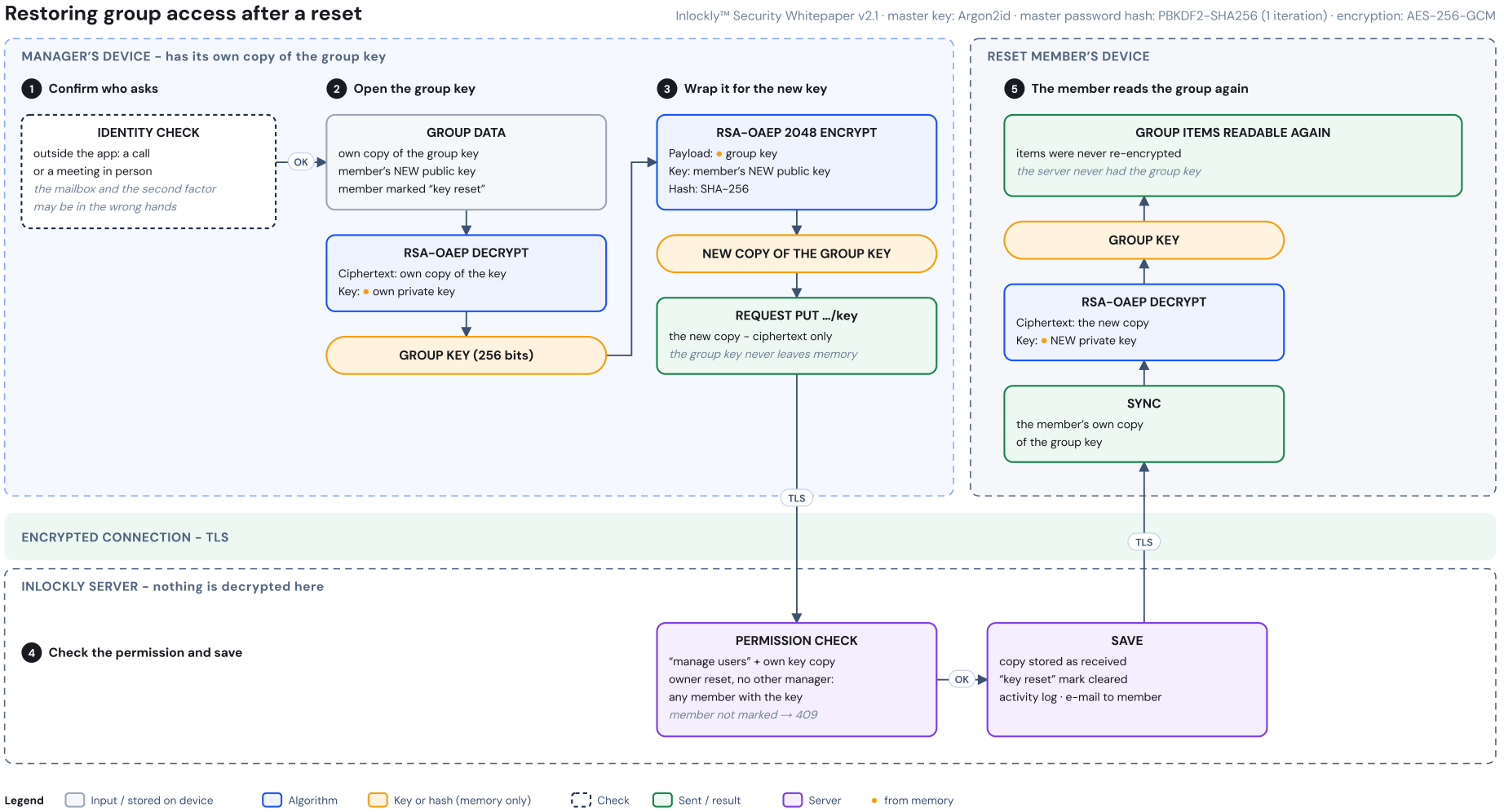


Figure 8. Restoring access to a group: a manager encrypts the group key with the member's new Public Key.

16.2 Restoring access, step by step

1. **Identity confirmation.** Before restoring access, the manager should confirm the person's identity outside the app - by phone or in person, through a channel the organization already knows. Someone who has taken over the email inbox and the second factor can look like the legitimate account owner to the service. The waiting period and notifications reduce this risk, but don't eliminate it.
2. **Opening the group key.** The manager's app decrypts (RSA-OAEP) their own copy of the group key with their Private Key.
3. **Encrypting for the new key.** The group key is encrypted (RSA-OAEP 2048, SHA-256) with the member's **new** Public Key. The resulting copy is sent to the server - the group key never leaves memory.
4. **Permission check and storage.** The server checks that the person restoring access has the user management permission and their own copy of the key, and that the member really is waiting for restored access (this isn't a way to swap keys). It stores the copy unchanged, clears the flag, records the operation in the group's activity log, and emails the member.
5. **The member can read the group again.** During sync, the member's app decrypts the new copy with their new Private Key. Group items weren't re-encrypted - they're readable right away.

⚠ Exception: when the group owner resets

If the group's **owner** has reset their account and no other member with the key has the user management permission, access can be restored to the owner by **any confirmed member who has the group key**. Without this, the group would be left without a manager forever (only the owner can transfer ownership). Account for this in your organization's procedures. If the person who reset was the only member with the group key, the group's data can no longer be read.

17

Comparing access recovery paths

Situation	What to do	Private vault	Groups	Required
You remember your Master Password but lost the phone with your 2FA app	Sign in with a backup code	Kept	Kept	Password + backup code
You forgot your Master Password but have your recovery key	Recover with the key	Kept	Kept	Key + email code + 2FA (if turned on)
You have neither your password nor your recovery key	Account reset	Deleted	After a manager restores access	Email code + 2FA (if turned on) + 7 days
You lost your password, your key, and your second factor	No self-service path	-	-	-

Why is there no way for a company to recover an employee's account?

Some password managers let an organization recover an employee's account, which means the administrator is technically able to read their private vault. In Inlockly™, we deliberately chose not to do that: an administrator can only restore access to **group** data the user was entitled to anyway. The private vault stays private - including from an employer.

18

Data sharing

Inlockly™ protects not only an individual user's data, but also the exchange of sensitive information with others. Instead of sending passwords in messages, users share them in shared group vaults or with an encrypted link that expires.

18.1 Group types

Group	For whom	Key features
Family	Loved ones, up to 5 people	A shared vault; everyone also has their own private vault
Team	Small teams, 5 to 10 seats	Folders, permissions, activity log
Company	Organizations, with no seat limit	Departments, folder permissions, roles, an extended log

Every group member keeps their own private vault, which other members and administrators can't access. Members of a company group can also create their own family group.

18.2 Encryption in groups

A group vault is encrypted with the group's own random 256-bit key. When a new person joins the group, a member with the right permission approves them, and their app encrypts the group key with the new person's Public Key (RSA-OAEP) and sends only that encrypted copy. The server never knows the group key.

Permissions (who can see which folder, who can edit, share, or manage members) are enforced by the server. The group is always determined on the server side from the resource the request acts on - a permission in one group doesn't allow operations in another.

18.3 Joining groups

You join a group with a join code or a link from an invitation emailed by a member with the right permission. Codes and invitations are valid for a limited time. Joining requires approval - only then does the new member receive the encrypted copy of the group key.

18.4 Sharing via link

A single item can be shared with someone outside your account using an encrypted link (Personal plan and up):

- the item's content is encrypted (AES-256-GCM) with **a new random key** just for that link;
- the key is part of the address **after the `#` sign** - browsers never send that part to the server, so the server only stores ciphertext;
- the link lasts from one hour to 30 days, or is **one-time** - it expires after it's first opened;
- access can be limited to specific email addresses - the recipient then confirms their address with a code;
- once the link expires or is used up, the server deletes the ciphertext, and only a record of the link (without content) stays in the history; you can also revoke a link at any time.

18.5 Vault backup

Exporting to an `.inlockly` file creates an encrypted backup with **a separate password**: the key is derived with Argon2id using a random salt and parameters stored in the file, and the content is encrypted with AES-256-GCM. The backup opens independently of the account. A wrong password and a corrupted file are indistinguishable.

> ⚠ Important
>
> Exporting to an unencrypted file (such as CSV) includes all your passwords in plain text. Use it only to move your data, and delete the file right after importing.

19

Helper features and privacy

Some app features need data from the vault. We design them so that this data never leaves the device.

19.1 SafetyMeter™

SafetyMeter™ rates password strength and points out reused passwords, websites without an encrypted connection, and accounts where it's worth turning on 2FA. Checking passwords against known breaches happens **on the device**: the app compares passwords with a bundled list of about 100,000 passwords from public breaches. We don't send passwords or their hashes to outside services. The results are calculated in memory and never reach the server.

19.2 2FA codes in items

A 2FA key saved with a login is part of the item's encrypted content. The one-time code (RFC 6238) is calculated by the app on the device - the server knows neither the key nor the codes.

19.3 Autofill

The browser extension and system autofill on phones fill in forms locally. Autofill settings for individual websites are encrypted with the Symmetric Key. System autofill on a phone unlocks the vault with the same Argon2id function as the app.

19.4 Website icons

The app shows website icons next to items. It fetches them through an Inlockly™ proxy server instead of directly from outside services, so the list of your websites doesn't reach third parties. The proxy only learns the name of the website whose icon the app is requesting. Icons are cached on the device, and you can turn them off completely in your preferences - then the app sends no requests at all.

© 2026 Inlockly™ · inlockly.com

19.5 Item change history

Previous versions of an item are stored encrypted with the same key as the item. The server only knows the time and author of each change, which are needed for sync and the activity log. How long history is kept depends on the plan.

20

Threat model and the limits of protection

No system protects against everything. Below, we describe what Inlockly™ protects against, and in which situations protection depends on the user and their device.

20.1 What Inlockly™ protects against

Threat	Why your data stays safe
A leak of the server's database or backups	The database contains ciphertext and BCRYPT hashes; keys are derived only from the Master Password or the recovery key
A compromised server or a dishonest employee	The server doesn't know the keys; it could block the service, but it can't read vaults
Eavesdropping on the connection	TLS, and beneath it, data already encrypted on the device
Tampering with encrypted data	The GCM authentication tag - modified ciphertext won't be accepted
Guessing the Master Password from stolen data	Argon2id (64 MiB per attempt) and BCRYPT on the server
Signing in with someone else's password	A second factor, new device alerts, attempt limits
Checking whether someone has an account	The same responses and timing for existing and nonexistent addresses
Bypassing 2FA with a database write	An HMAC seal on everything that determines access

Threat	Why your data stays safe
Taking over an inbox and second factor to force a reset	A 7-day waiting period, daily notifications, and one-click cancellation

20.2 The limits of protection

- **An infected device.** Malware with full access to an unlocked device (for example, a keylogger) can capture the Master Password or read data from an unlocked vault. Keep your system up to date, and don't install software from untrusted sources.
- **A weak or reused Master Password.** Argon2id raises the cost of an attack, but it's no substitute for a good password.
- **Phishing.** A fake site can trick you into entering your Master Password. The Inlockly™ extension only suggests data on the correct domain - if no suggestion appears, check the address.
- **Losing every factor.** Without the Master Password, the recovery key, and the second factor, there's no self-service path to recover the account.
- **Metadata.** The server knows the administrative information described in the "What the server sees" chapter - including group and folder names, the time of changes, and device IP addresses.
- **People with access to a group.** A group member can read and copy the data they have access to. Revoking access doesn't delete copies they've already made - after someone leaves, change the passwords that were shared with them.
- **The account's TOTP secret.** A second factor in an authenticator app requires the server to know its secret (encrypted with a server key). This doesn't affect the confidentiality of the vault.

21

Recommendations

21.1 For users

- Choose a long, unique Master Password and don't use it anywhere else.
- Keep your recovery key outside Inlockly™ - printed, or on a storage device somewhere safe.
- Turn on two-factor authentication - ideally an authenticator app or a YubiKey - and store your backup codes independently of your phone.
- Respond to security emails: a new sign-in, a reset request, a backup code being used. If it wasn't you, cancel the request and change your Master Password.
- Set a short auto-lock timeout on devices that other people also use.
- Check SafetyMeter™ regularly and change weak, reused, and breached passwords.

21.2 For group and company administrators

- Create a procedure for **confirming identity** before restoring access after an account reset - outside the app, through a channel you already know.
- Make sure **at least two people** have the user management permission and a copy of the group key. Resetting the only manager makes restoring access harder, and resetting the only person with the key means losing the group's data.
- Grant permissions on a need-to-know basis: folder and department access only for those who need it.
- Don't put sensitive information in the names of groups, departments, and folders - they aren't encrypted.
- When an employee leaves, remove them from the group and change the passwords that were shared with them.
- Review the group's activity log, especially after a member's account reset.
- Remind members to keep their recovery key - it's the simplest way to avoid losing data.

22

Glossary

Term	Meaning
Master Password	A password known only to the user; it never reaches the server
Master Key	A 256-bit key derived from the Master Password with Argon2id; kept only in the device's memory
Master Password Hash	A one-way hash of the Master Key sent to the server for authentication
Symmetric Key	A random 256-bit account key; it encrypts the private vault and the Private Key
Protected Symmetric Key	The Symmetric Key encrypted with the Master Key; stored on the server
RSA Key Pair	A Public Key (public) and a Private Key; used to pass along group keys
Protected Private Key	The Private Key encrypted with the Symmetric Key; stored on the server
Group key	A random 256-bit group vault key; each member has their own encrypted copy
Recovery key	128 random bits (28 characters) for recovering an account after forgetting the Master Password
Wrap key / auth key	Keys derived from the recovery key: the first encrypts a copy of the Symmetric Key, the second is proof for the server

© 2026 Inlockly™ · inlockly.com

Term	Meaning
Verifier	A value that lets the server recognize a correct recovery key without knowing the key
Vault	A set of items encrypted with the same key: private or group
Item	A single entry in the vault: a login, card, note, and so on
Nonce (initialization vector)	A random, single-use value used for every AES-GCM encryption
Authentication tag	A 128-bit GCM value that detects any change to the ciphertext
HMAC seal	A server signature that protects access-determining data from modification in the database
Session token	An identifier for a signed-in device
Trust token	Skips the 2FA step on a trusted device
Waiting period	The 7 days between an account reset request and the ability to complete it
CSPRNG	A cryptographically secure random number generator
Zero-knowledge	A model in which the service provider has no keys to the user’s data

© 2026 Inlockly™ · inlockly.com

23

Standards and sources

- NIST FIPS PUB 197 - Advanced Encryption Standard (AES), https://csrc.nist.gov/pubs/fips/197/final
- NIST SP 800-38D - Galois/Counter Mode (GCM) and GMAC, https://csrc.nist.gov/pubs/sp/800/38/d/final
- ISO/IEC 18033-3 - Encryption algorithms, Part 3: Block ciphers
- IETF RFC 9106 - Argon2 Memory-Hard Function for Password Hashing, https://www.rfc-editor.org/rfc/rfc9106
- IETF RFC 8018 - PKCS #5: Password-Based Cryptography (PBKDF2), https://www.rfc-editor.org/rfc/rfc8018
- IETF RFC 5869 - HMAC-based Extract-and-Expand Key Derivation Function (HKDF), https://www.rfc-editor.org/rfc/rfc5869
- IETF RFC 2104 - HMAC: Keyed-Hashing for Message Authentication, https://www.rfc-editor.org/rfc/rfc2104
- IETF RFC 8017 - PKCS #1: RSA Cryptography Specifications v2.2, https://www.rfc-editor.org/rfc/rfc8017
- IETF RFC 6238 - TOTP: Time-Based One-Time Password Algorithm, https://www.rfc-editor.org/rfc/rfc6238
- IETF RFC 8446 - The Transport Layer Security (TLS) Protocol Version 1.3, https://www.rfc-editor.org/rfc/rfc8446
- Crockford's Base32, https://www.crockford.com/base32.html
- N. Provos, D. Mazières, "A Future-Adaptable Password Scheme" (BCRYPT), USENIX 1999
- OWASP Password Storage Cheat Sheet, https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html
- NCSC, "Most hacked passwords revealed as UK cyber survey exposes gaps in online security", https://www.ncsc.gov.uk/news/most-hacked-passwords-revealed-as-uk-cyber-survey-exposes-gaps-in-online-security
- W3C Web Cryptography API, https://www.w3.org/TR/WebCryptoAPI/

© 2026 Inlockly™ · inlockly.com

24

Changelog

24.1 v2.1 - October 5, 2026

This whitepaper describes new access recovery paths and has a new format: a web version and a PDF generated from the same content. The fundamental data protection principles and the cryptographic scheme from version 2.0 remain unchanged.

- **Recovery key** (new chapter and diagram): a 128-bit key created at sign-up lets you recover your account after forgetting your Master Password without losing data. We removed statements saying access couldn't be recovered.
- **Account reset without a recovery key** (new chapter and diagram): a 7-day waiting period with notifications and the option to cancel; the private vault is deleted, while the account, plan, and memberships stay.
- **Restoring access to groups after a reset** (new chapter and diagram) by a group manager, along with the identity confirmation principle and the exception for the group owner.
- **Two-factor authentication:** added authenticator apps (TOTP) and 10 backup codes, plus a description of attempt limits, replay protection, and method availability by plan.
- The account creation diagram now includes recovery key generation; the sign-in descriptions are updated (session and trust tokens under an HMAC seal, cancellation of a pending reset).
- New chapters: "Key points at a glance," "What the server sees - and what it doesn't," a summary of cryptographic parameters and the key hierarchy, sharing via link and vault backup, helper features and privacy, the threat model, recommendations for users and administrators, a glossary, and standards and sources.
- The infrastructure chapter is limited to the current state (servers in the European Union); we'll publish a detailed description of the architecture in the next version.
- Diagrams in Polish and English, generated from a single model.

24.2 v2.0 - September 24, 2026

The second version of the Inlockly™ Security Whitepaper. The fundamental data protection principles remain unchanged, but their cryptographic implementation has changed. The new scheme isn't compatible with the one described in version 1.0.

- We derive the Master Key with Argon2id (64 MiB, 4 iterations, 4 threads) instead of PBKDF2-SHA256. The salt is the SHA-256 hash of the email address. PBKDF2 is now used only to derive the Master Password Hash, in a single iteration.
- We encrypt data and keys with AES-256-GCM, which detects any modification of encrypted data. A new initialization vector is generated for every encryption.
- The Symmetric Key is 256 bits long.
- We protect copies of group vault keys with RSA-OAEP (2048-bit, SHA-256).
- The default inactivity period after which vaults lock is 5 minutes.
- We updated the description of the technology (TLS instead of SSL) and of data protection regulations, removed the description of regenerating the Symmetric Key, and fixed editorial errors.

24.3 v1.0 - January 13, 2024

The first public release of the Inlockly™ Security Whitepaper. This document helps you understand how we implemented every solution in Inlockly™, and how we take care of your data and sensitive information.