heylogin Security Whitepaper

Deep dive into our end-to-end encryption & protocols

heylogin GmbH

April 29, 2026, Version: v3.7

Contents

1 Why legacy Password Managers are insecure

Remembering only one password, the *Master Password* instead of many passwords is the main feature of legacy Password Managers. With heylogin, a Master Password is no longer necessary. Instead, it uses the secure element present in modern smartphones and replaces the Master Password with *Swipe to Login*. Secure elements are security chips that protect secrets against unauthorized access and brute-force attacks. When authorization is required, the user is asked to swipe on their smartphone instead of entering a Master Password. This dependency also makes heylogin two-factor secure by design because login requests have to be authorized on a second device: the smartphone. While heylogin's Swipe to Login technique feels like an authentication method, it actually uses end-to-end encryption from the smartphone to the browser to make passwords available.

Before going into the details of heylogin's security architecture, we'll briefly explain the usability and security issues of legacy Password Managers and their use of Master Passwords.

1.1 Usability issues of legacy Password Managers

Legacy Password Managers require users to remember and regularly enter a Master Password. This Master Password is used to protect private information, such as usernames and passwords. It is well studied that users, especially non IT-savvy people, have usability problems with Master Passwords and their handling.

- **Re-using Master Password:** Studies show that not all users are able to generate and remember a sufficiently secure Master Password. In a study from 2023, 28% of all participants re-use their Password Manager's master password as a password on a website account[1]. In another study by Pearman et al.[2], participants reused a different password as their Master Password or had it generated on a website. The participants involved had no technological training. So, especially for people who are not IT experts, using a Password Manager with a Master Password can actually reduce their security to a single point of failure.
- **Time wasted:** Depending on the implementation and the security policies used, the Master Password must be retyped regularly by the user in order to temporarily decrypt the vault. We assume about 1 hour / month / user, which are spent for the regular typing of the Master Password and the password management.

[1] Security.org Team, Password Manager Industry Report and Market Outlook (2023-2024), https://www.security.org/digital-safety/password-manager-annual-report/

[2] Pearman et al., Why people (don't) use password managers effectively, 2019

- **Bad user acceptance:** Employees forget their Master Password, which leads to support tickets and bad user acceptance.

The use of legacy Password Managers is thus mainly associated with annoyances that go beyond the normal conflict between security and user-friendliness. Existing solutions cannot easily change their security architecture because basic user flows and user expectations go hand in hand with the Master Password.

1.2 Security issues of legacy Password Managers

Technically, the Master Password is used to encrypt and decrypt all stored information. A Master Password must be complex and kept private, as it is the single secret to all information. There are several problems associated with this cryptographic design:

- **1-factor security:** While many Password Managers allow the setup of another factor, such as TOTP, U2F or FIDO2/WebAuthn. Due to it's usability challenges, these 2-factor mechanisms are not enabled by default and thus not used by regular users. The Master Password remains as the only factor protecting the vaults. Since Master Passwords are chosen by humans, they are not perfectly random and almost always contain deterministic pattern.
- **Offline brute-force attacks:** The Master Password, as a factor of knowledge, cannot be protected against brute-force attacks as soon as they are performed offline. When a password vault is stolen or a data leak occurs at the large commercial Password Managers, the encrypted vaults can be attacked "offline", i.e., there is no interactive protocol involved that rate-limits retries. A brute-force attack or dictionary attack is only slowed down by the vault's Key Stretching function. However, this never achieves the protection of a Hardware Security Module (HSM) since Key Stretching functions only slow down the brute-force attack, but can never limit the number of tries like a HSM could.

1.3 Architecture of legacy Password Managers

For a deeper understanding of the security issues, it first helps to understand the basic architecture of legacy Password Managers.

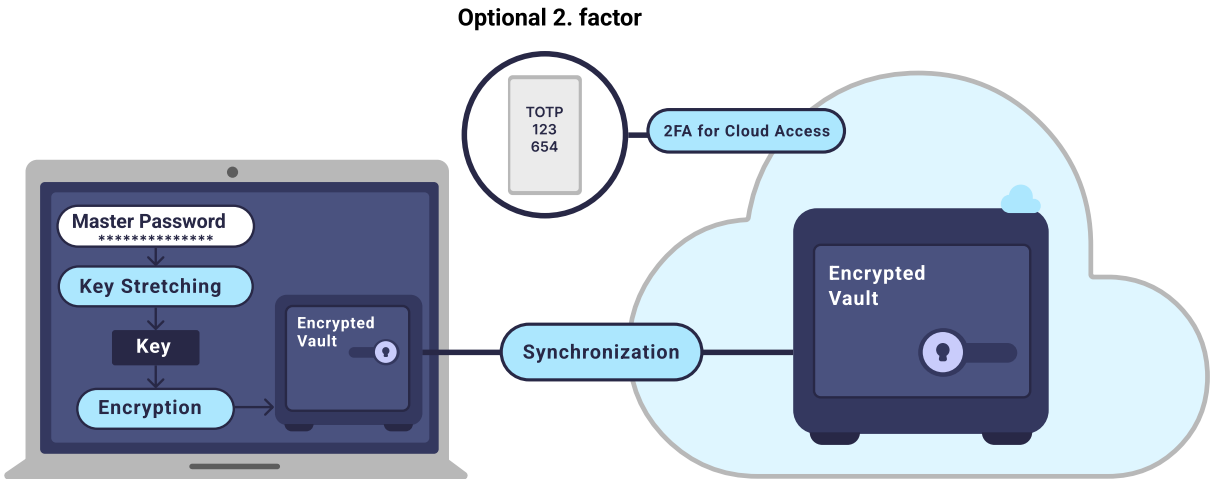

Figure 1: Highly simplified security architecture of legacy Password Managers

The figure shows a highly simplified version of the architecture of legacy Password Managers. Here, a *Master Password* is chosen by the user. From this *Master Password* a *Key* is derived by using a *Key Stretching* function, which typically involves several iterations. This *Key* is then used for the *Encryption* of the *Vault*. This *Encrypted Vault* is synchronized with the cloud.

2-factor authentication mechanisms, which can be enabled optionally, are an additional protection when accessing the cloud infrastructure. As shown in the figure, the 2-factor authentication is not part of the vault encryption process.

1.4 Attacking legacy Password Managers

As always with software architecture, there are a lot of ways to attack it. Without any doubt, the end-to-end encryption of private data, such as usernames and passwords, is the main security guarantee a password manager provides. Thus, we focus on attacks breaking the end-to-end encryption.

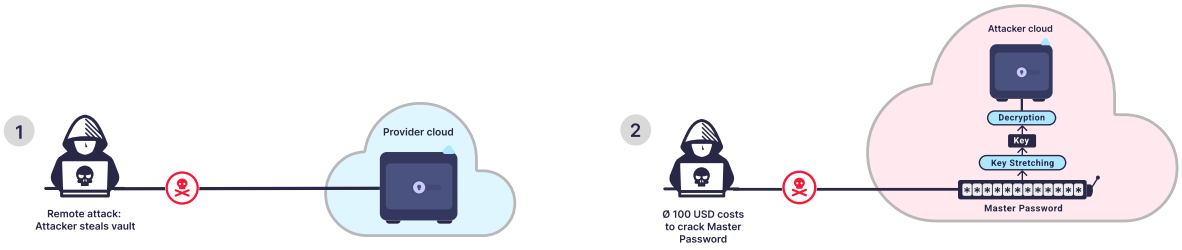

Figure 2: To execute an offline brute-force attack, 1) the attacker steals customer vaults and then 2) brute-forces possible Master Passwords until the correct one is found.

In step 1, we consider an attacker who stole the encrypted vaults. This basic premise is in no way different to other data breaches, which happen daily on the Internet. The most famous data breach that specifically happened to a Password Management provider, happend to LastPass in December 2022. The attackers gained access to the cloud infrastructure and bypassed the protective mechanisms of a single LastPass employee. This enabled them to steal all customer vaults[3].

In step 2, the stolen encrypted vault is uploaded to a private cloud of the attacker to perform the actual offline brute-force attack. The attacker can now try through an extremely large number of possible Master Passwords automatically. For this, they execute the same steps as in the original architecture: Use a *Master Password* to derive a *Key* using *Key Stretching* and then trying the *Decryption* of the vault. If decryption fails, try the next *Master Password*. In case of LastPass, it costs on average 100 USD of cloud computing power to crack a 12-character long Master Password.

In this attack, two things are important to note:

- **2-factor authentication is useless:** Even if the customer enabled 2-factor authentication in their Password Manager, it did not provide any additional protection. Actually, when stealing the vaults of all customers, probably only one 2-factor authentication of one employee of the Password Management provider was circumvented. This is typically done by executing a highly targeted and costly attack that exploits vulnerabilities on an employee device of the Password Management provider. After the successful theft of the customer vaults, all 2-factor authentication mechanisms of the customers are no longer useful since they are not part of the encryption process.
- **Key stretching is a race to the bottom:** The only thing that slows down the brute-force attack is the key stretching function. This is what these functions were designed for, but

[3]LastPass, Notice of Recent Security Incident, 2022, https://blog.lastpass.com/2022/12/notice-of-recent-security-incident/

they have to be adapted regularly because the attackers also have better and better hardware. Most legacy Password Managers haven't adapted their configuration and iteration count for several years. They neglected the only protection against attacks they had and only acted after the LastPass attack went public. At that point, they should have at least used 6-times larger iterations counts. Even this may not protect against specialized attacker hardware. A better decision would be to change their cryptographic architecture to use memory-hard key stretching algorithms, such as Argon2, which has already been standardized by cryptographers in 2015.

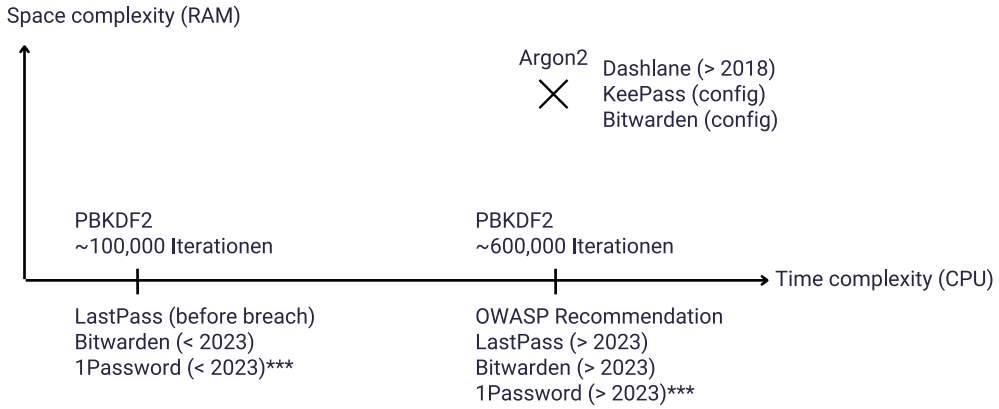

Figure 3: A simplified comparison of key stretching functions and iteration counts used by legacy password managers. Note: 1Password's Secret Key protects against brute-force attacks, but only if it hasn't been stored digitally.

2 How heylogin solves these issues

At heylogin, we designed a passwordless Password Manager. Our solution consists of a vastly different user experience paired with a modern hardware-based end-to-end encryption architecture.

2.1 Security chip replaces Master Password

We completely remove the need for a Master Password. Instead, we generate a cryptographic seed directly in the security chips of the respective user devices to derive the specific keys for end-to-end encryption. In contrast to a Master Password, a seed is 256 bit secure, completely random and contains no human patterns.

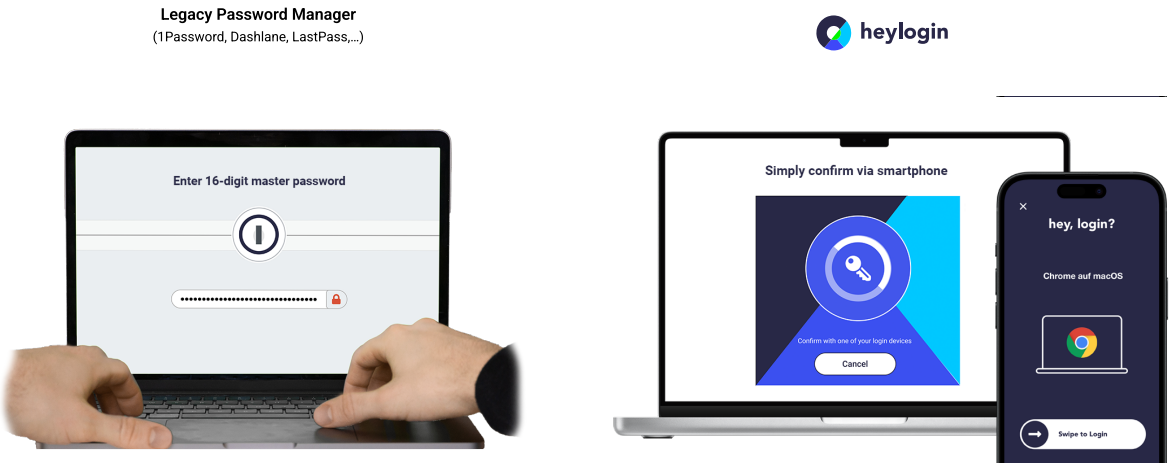

Figure 4: Employees can conveniently confirm their login process via a swipe-to-login function in our iOS or Android app.

heylogin supports security chips from different users devices. Typically, it uses the security chips built into smartphones on iOS and Android. This also allows the usage of smartwatches to unlock heylogin if they are properly connected over Bluetooth with the phone. In this case heylogin still uses the security chip inside the smartphone.

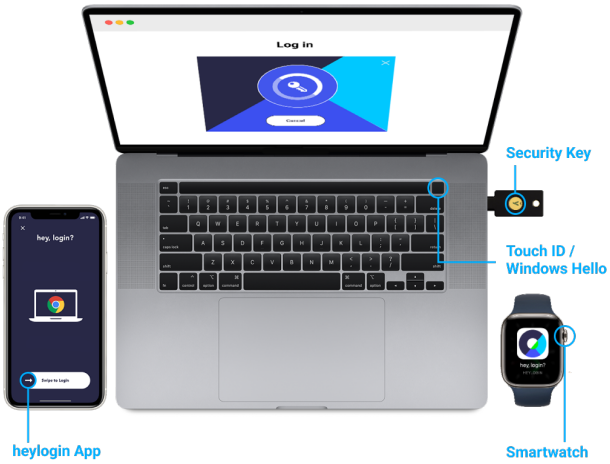

Figure 5: We support different login devices for unlocking heylogin

As an alternative, FIDO2 security keys can be used, which are connected over USB or NFC. With these devices, the security chip inside these security keys can be used as an alternative to the user's phone or even as a standalone device without a phone. Security chips in PCs and Macs offer another way to unlock heylogin, via the Windows Hello and Touch ID APIs.

2.2 1-click login automation

heylogin's browser extension automates the login experience by detecting login forms, e.g. username and password fields, and then showing an overlay on top. This hides the inconveniences of the login process and instead shows a 1-click button labeled with the user's website account.

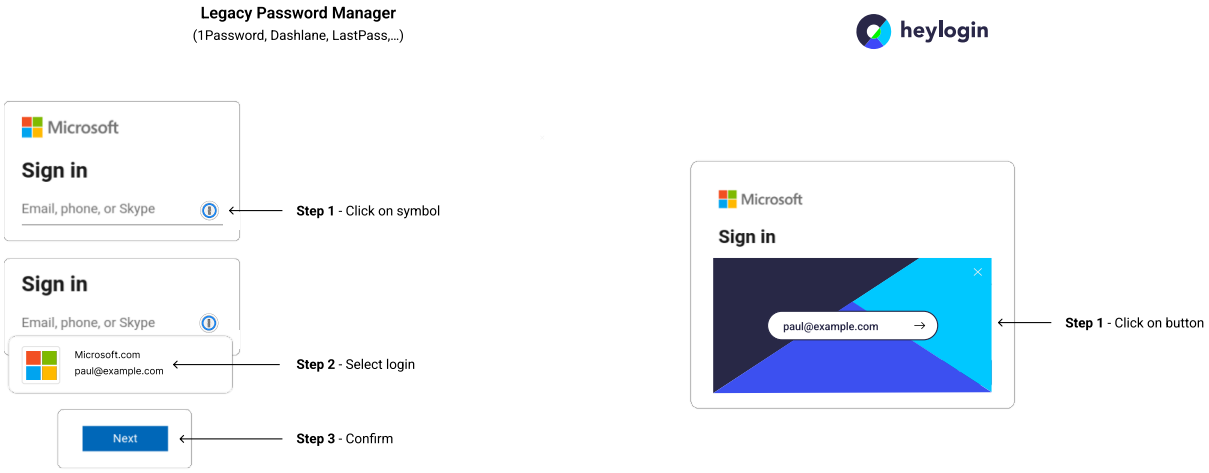

Figure 6: Simply log in with one click instead of laboriously selecting the right account from a drop-down menu. This reduces the mental load when browsing and makes logging in extremely easy, even for non-IT employees.

The heylogin overlay works with all websites and even two-step logins, where username and password are on two different sites are also supported as well as automated entry of TOTP codes. The algorithm has been fine tuned over several years and is continuously updated.

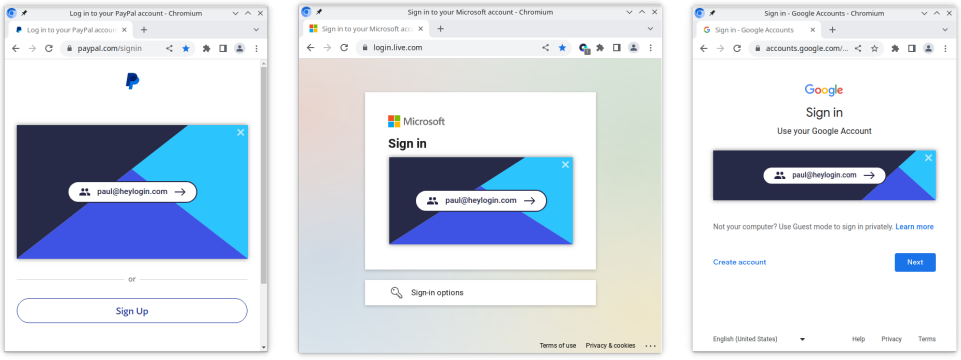

Figure 7: Our algorithm automatically detects login forms on all websites and shows the heylogin overlay on top.

2.3 heylogin user experience

The resulting experience feels like a Single-Sign On (SSO) but is technically a Password Manager.

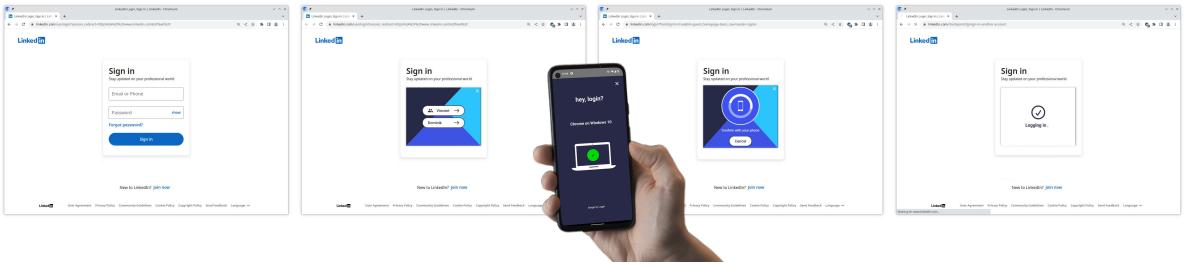

Figure 8: The full heylogin experience using the example of linkedin.com

1. In milliseconds, the overlay algorithm detects the login form on the website.
2. The browser extension renders the overlay precisely on top of the login form.
3. After clicking the button labeled with the user's website account, a notification is shown on the smartphone.
4. After unlocking the security chip locally with Face Unlock, Fingerprint or PIN, the browser extension is unlocked and can now access private data, such as username and passwords, inside the vault. Username and password is automatically filled in behind the overlay to automate the website login process.

2.4 Real 2-factor security

For multi-factor authentication / security, three factors are typically discussed: knowledge ("something only the user knows", such as PIN, password, master password), possession ("something only the user has", such as security chips, certificates files) and biometric factors ("something the user is", such as fingerprint, face unlock).

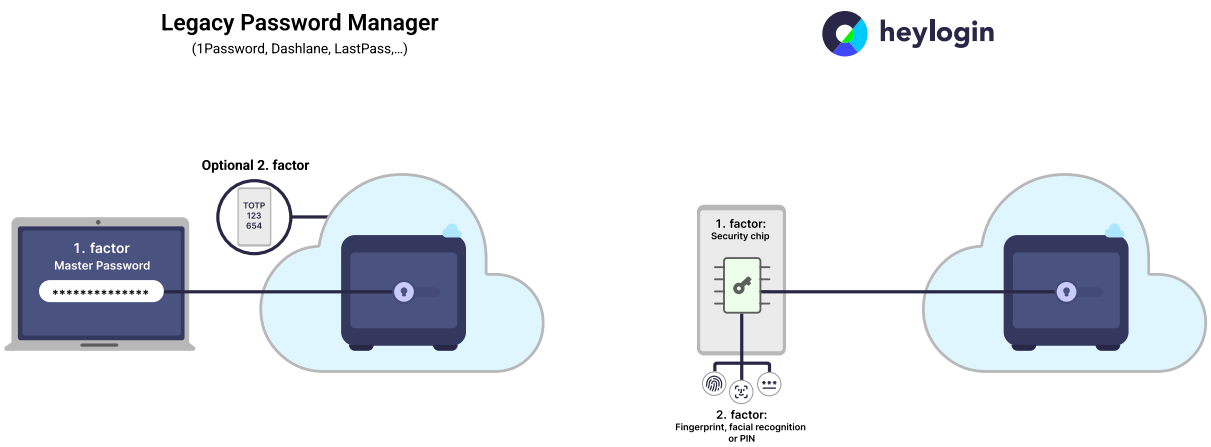

Figure 9: With legacy Password Managers, the 2nd factor is not activated by default and is not part of the vault encryption, but only an authentication to the cloud. With heylogin, the 2nd factor is active by default and part of the encryption.

As explained before, legacy Password Managers use the Master Password as a 1st factor (knowledge factor) and optionally implement 2nd factors, such as TOTP codes (knowledge factor) or security keys (possession factor). These 2nd factors, are only authentication mechanisms for an additional protection when accessing the cloud infrastructure, but not part of the vault encryption process. In case of an infrastructure breach, this means that they effectively only have 1-factor security with a master password.

In contrast, heylogin has real 2-factor security as part of the vault encryption process. heylogin's 1st factor is always a security chips, built inside the user's smartphone, FIDO2 security key or TPM (possesion factor). heylogin's 2nd factor is the factor used to unlock the security chips. In case of smartphones, this is typically fingerprint (biometric factor), face unlock (biometric factor) or PIN (knowledge factor). The 2nd factor always works on the login device itself locally and no biometric or PIN data is transmitted to our servers.

While Master Passwords and PINs are both knowledge factors, there is a crucial difference how they work. As shown before, in a offline brute-force attack, Master Passwords can be guessed with unlimited attempts, since these attacks are only slowed down by the key derivation method. In contrast, PIN attempts are strictly limited by the security chip hardware. An attacker must therefore not only physically steal the smartphone (1st factor), but also unlock the security chip with a 2nd factor. It is important to remember that the use of the security chip ensures that our cryptographic seed can never be accessed by bypassing the phone's authentication mechanisms, e.g., by prying out the storage and reading it through an external reader.

3 Core security design

Logins are encrypted in strictly separated vaults with the respective security chips. This allows normal users of an organisation to access their personal vault and all team vaults for which they have permissions.

In addition, all vaults within an organisation are also encrypted with the security chips of the admins. This makes it possible to cryptographically restore an account if a device is lost or needs to be replaced. In terms of security, however, admins do not have access to the passwords in the personal vaults.

All access, whether by admins or users, is end-to-end encrypted and requires the respective security chip for decryption. Our platform offers a strict security architecture where we as heylogin GmbH have no access to your logins.

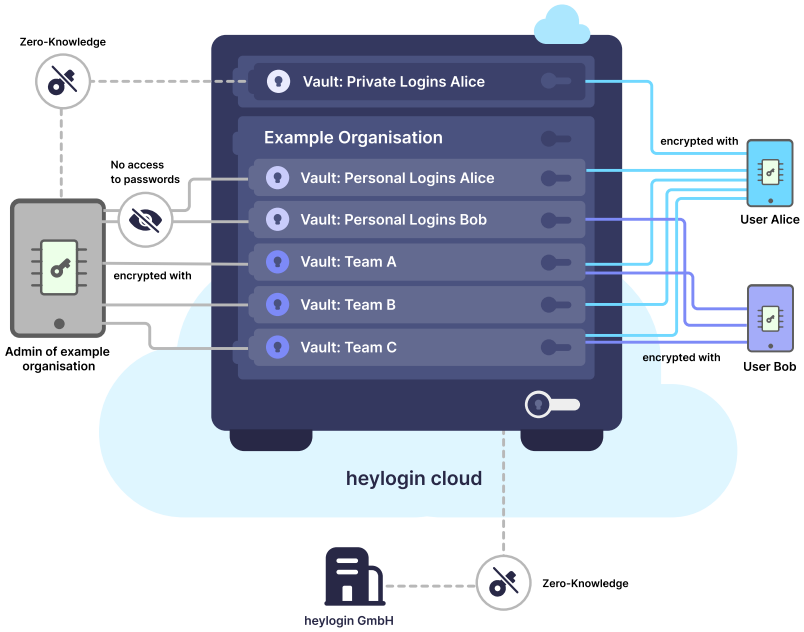

Figure 10: Overview over heylogin's security architecture

3.1 Synchronization and unlock process

In the following, we describe heylogin's pairing, synchronization and vault unlocking on a high level for a typical setup between smartphone and browser extension. This section only provides a simplified view. In reality, more cryptographic keys and vaults are involved. For a detailed view,

please consult the later sections about *heylogin's cryptographic architecture*.

3.1.1 Pairing app with browser extension

After installing the app, a pairing process is initiated by scanning a QR code. The QR code enables a one-time, out-of-band authenticated key exchange between the browser and the app. The browser shows a QR Code containing an ephemeral public key; your phone encrypts its authenticator seed to this public key and sends it to the browser. The browser decrypts the seed, dervies the login key pair, creates its own session key pair, and becomes a paired device. The server only relays encrypted data and never sees any secrets.

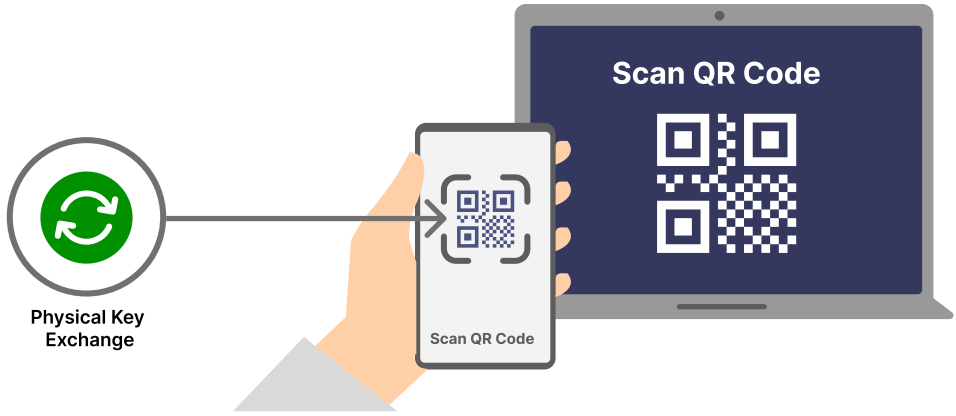

Figure 11: Out-of-band authenticated key exchange by physically scanning a QR code

3.1.2 Synchronisation of the encrypted vaults

All vaults belonging to a user are automatically synchronized in the background across all their devices. This includes the heylogin app on mobile devices as well as browser extensions. To keep everything up to date in real time, all clients maintain a streaming connection to the heylogin cloud. This means that whenever a change is made, such as adding, editing, or deleting a login, all paired devices receive the update immediately and automatically without requiring manual refresh.

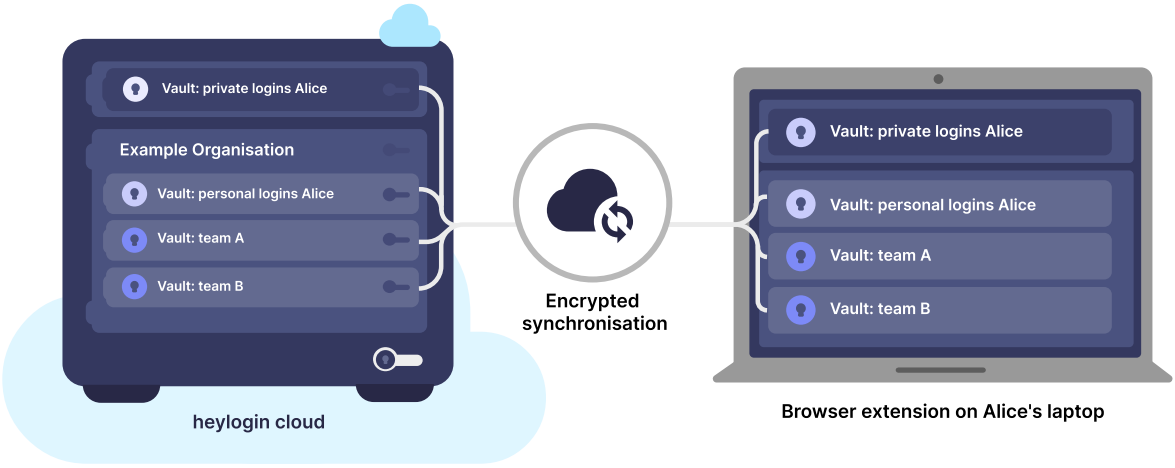

Figure 12: Synchronisation of the encrypted vaults

Importantly, the heylogin cloud acts solely as a transport and storage layer. It has no ability to decrypt any data. All vault content is end-to-end encrypted before it leaves the user's device, and decryption keys are only available locally, secured by the device's hardware (e.g., security chip or enclave). This ensures that even if the server infrastructure were compromised, attackers would gain access only to encrypted data with no way to unlock it. In this architecture, trust is placed in the user's own devices, not in the cloud. Synchronization is fast, seamless, and secure by design.

3.1.3 Unlocking the security chip

If a browser extension needs to decrypt login data, it does not have access to the cryptographic keys directly. Instead, it sends a request to the user's paired phone, which acts as the secure authenticator. Upon receiving the request, the heylogin app on the smartphone displays a prompt to the user. To proceed, the user must locally unlock the security chip on their phone using a second factor, typically a fingerprint, face unlock, or PIN, depending on the platform and device settings.

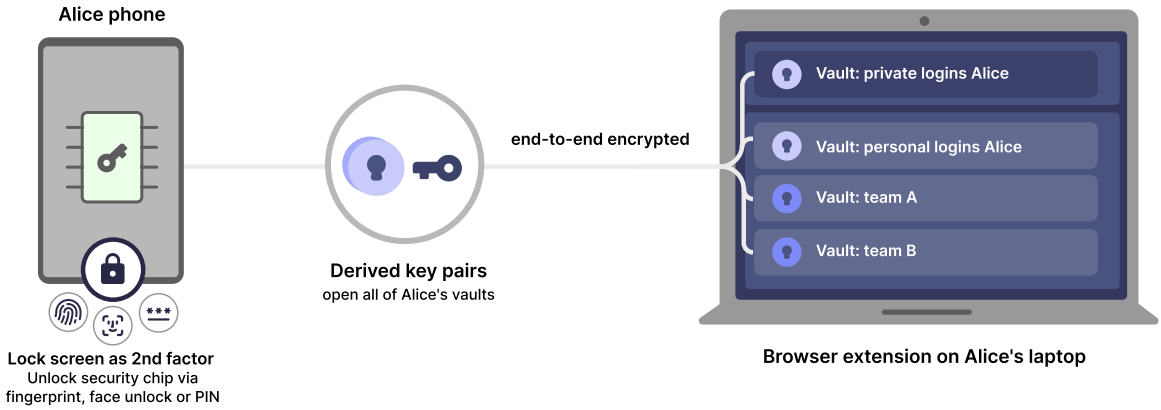

Figure 13: Unlocking the security chip with a 2nd factor

Once successfully unlocked, the device's authenticator seed is encrypted to the session's public key and stored on the server for the browser to retrieve (end-to-end encrypted between device and browser). The browser then derives all needed key pairs and discards the seed. The encrypted authenticator seed is not kept on the server forever, see *Locking and unlocking a device*.

3.1.4 Unlocking heylogin via vault decryption

heylogin's browser extension is unlocked by decrypting the user's synchronized vaults locally using the key pairs derived from the received authenticator seed. This includes all login entries, TOTP secrets, and any protected fields the user has access to. Once decrypted, the sensitive data becomes available in memory and can be used to automatically fill in login forms on websites, enabling a seamless 1-click login experience.

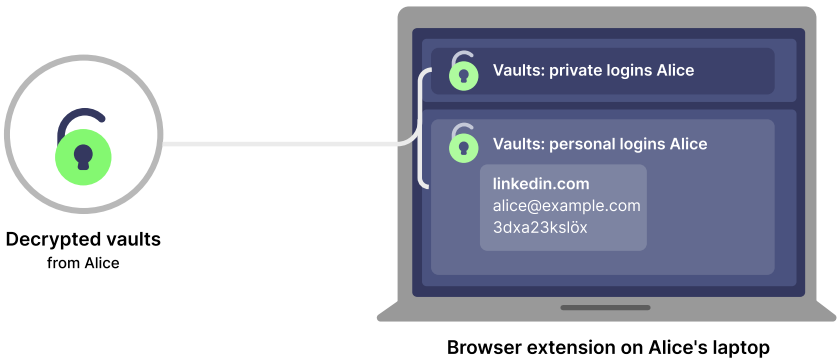

Figure 14: Unlocking heylogin decrypts the vaults and makes the logins available in-memory

Because the decrypted data exists only in the local memory of the extension and is never stored unencrypted, it maintains a high level of security even during use. Any actions the user takes, such as adding new logins, updating existing entries, or deleting outdated credentials, are encrypted again on the client side and immediately synchronized back to the cloud via the streaming connection. These updates are then automatically propagated to all other trusted devices connected to the user's account.

3.1.5 Locking heylogin again

heylogin does not stay unlocked indefinitely. After a configurable period without activity, heylogin locks itself and a new unlock via the smartphone is required. On locking, the derived keys and the decrypted data are discarded from memory.

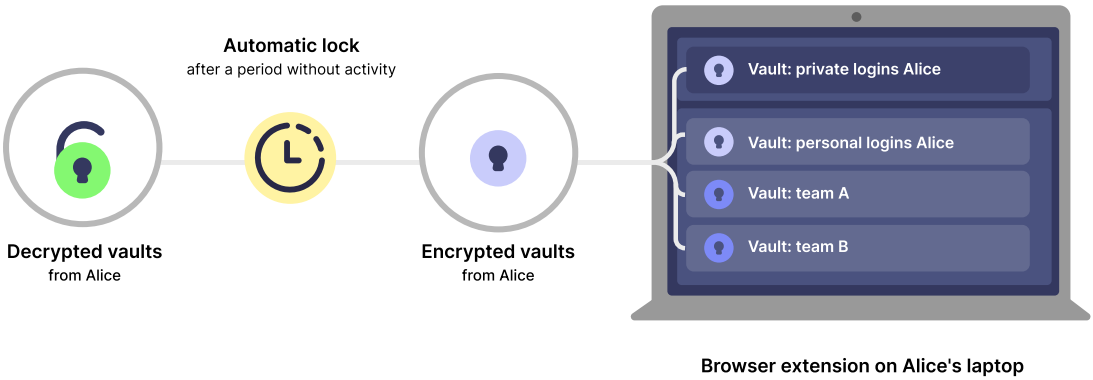

Figure 15: heylogin locks automatically after a period without activity, discarding all derived keys and decrypted data from memory

Activity is not limited to interactions with heylogin itself: any user-driven event in the browser,

such as scrolling, switching tabs, or typing into a form, keeps the session alive. The lock delay only starts to elapse once the browser has been idle continuously. Each user can choose the lock delay from a fixed set of options: 5 minutes, 15 minutes, 1 hour, 2 hours, or 8 hours. Organization admins can enforce a maximum lock delay for all members, capping the values users may select.

As an additional security measure, every paired device is forcibly locked once per day at 2:00 AM, based on the local clock of the smartphone that authorized the unlock. This nightly reset is enforced regardless of the chosen inactivity delay and ensures that a fresh unlock via the smartphone is required on the next day.

Beyond automatic locking, users can lock heylogin manually at any time, either from the smartphone app or by pressing “Log out” in the bottom-left corner on heylogin.app. This is useful when stepping away from a shared or unattended device.

3.2 What we encrypt and how

In heylogin, different categories of data are protected with different layers of encryption. Some information, like usernames or websites, needs to be shown in overlays and is therefore available on paired devices even when locked. More sensitive secrets, such as passwords and TOTP keys, are protected by a second layer of end-to-end encryption and only become accessible once a device is unlocked. The separation between these two encryption layers is described in detail in the *Pairing another device* section.

Certain operational metadata such as organization names, account emails, IP addresses, unlock times, activity times, and device data is secured with transport encryption (TLS) but is not end-to-end encrypted. For details about how this metadata is processed from a privacy and legal perspective, please refer to our Data Processing Agreement (DPA).

Encryption layer	Data types	Access	Algorithms
Transport secure (TLS)	Organization name, account emails, IP addresses, unlock times, activity times, device data	Visible to provider as required for account management, troubleshooting, and support	only TLS 1.2 or TLS 1.3 (AES-GCM or ChaCha20-Poly1305, ECDSA, HSTS, DNS CAA, Let’s Encrypt)

Encryption layer	Data types	Access	Algorithms
1st layer end-to-end encryption	Usernames (stored with logins), websites/domains, team names, unprotected custom fields	Available on all paired devices, even when locked (used for overlays)	XSalsa20-Poly1305, Curve25519 (TweetNaCl.js library[4])
2nd layer end-to-end encryption	Passwords, TOTP secrets, protected custom fields	Available only on paired and unlocked devices	XSalsa20-Poly1305, Curve25519 (TweetNaCl.js library)
Additional encryption at rest	Server backups	Provides additional protection by keeping server backups inaccessible even if backup storage is breached	ChaCha20-Poly1305 (age[5])

This layered design ensures that operational metadata remains minimal and secured in transit, while sensitive logins and secrets are strictly protected with end-to-end encryption. Backups gain an additional protection layer through encryption at rest.

4 Advanced security design

In addition to the core architecture, heylogin includes advanced features that each come with their own dedicated security design. These features are not merely usability enhancements but are built with the same end-to-end encryption principles as the core system.

4.1 TOTP automation

In modern authentication architectures, the classical combination of identifier (user name, email, etc.) plus (static) password is typically enhanced through the use of an additional changing one-time password that requires the authenticating party to not only have knowledge

[4] Dmitry Chestnykh, TweetNaCl.js, https://tweetnacl.js.org
[5] Filippo Valsorda, age file encryption, https://age-encryption.org/

of the static credentials but also knowledge of the secret generating the changing one-time password. A specific kind of one-time password are time-based one-time passwords (TOTP) that combine a static secret with the current time to generate a changing password, a TOTP code. TOTP codes are single use codes with a short expiry even when not used.

TOTP is commonly called a 2nd factor due to its supplemental usage in addition to the static credentials of identifier plus password. Traditionally, a separate application is used to manage and generate TOTP codes. heylogin supports storing TOTP secrets and generating codes directly with the login they belong to. This does not diminish the security of TOTP codes compared to a second application managing and generating TOTP codes.

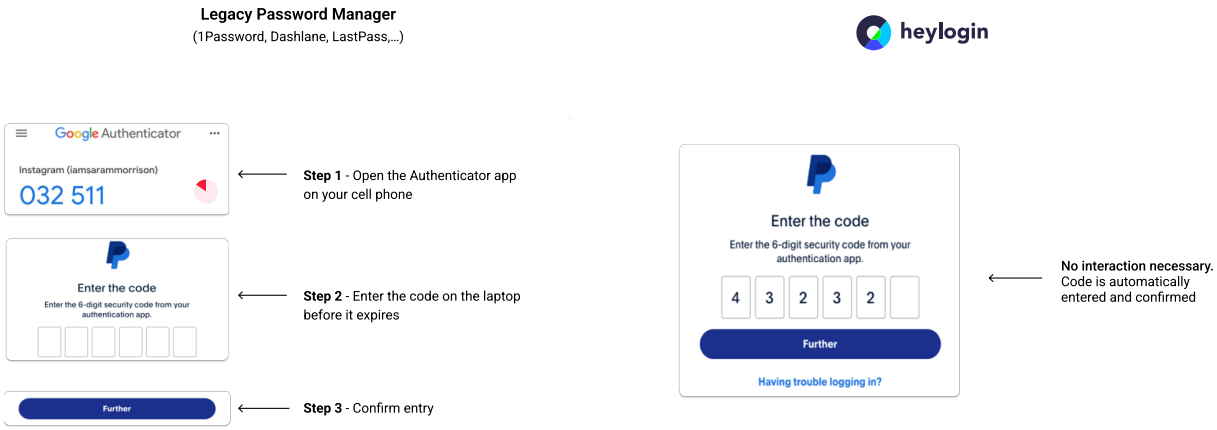

Figure 16: Thanks to the simple setup and automatic TOTP entry, typing is no longer necessary. So you can easily activate 2-factor for your important accounts, heylogin does the work for you.

With a standalone TOTP application, e.g., Google Authenticator, a user needs to manually enter the generated code when asked for it (**left picture**). The security of this system lies in the fact that the user not only needs something they know (identifier plus password) but also something they have (their smartphone with the TOTP application), which is their 2nd factor. With heylogin, both pieces of information now come from the smartphone application and allows heylogin to autofill TOTP codes (**right picture**). As heylogin enforces a successful unlock of the smartphone before allowing access to the stored logins, heylogin is also 2nd factor secure. You need access to the smartphone as the 1st factor (possession) and either something you know (a passcode or similar) or something you are (biometrics) as a 2nd factor. This is also true when using a hardware security key as heylogin enforces not only possession but, depending on the key used, a passcode or a biometric check to unlock the hardware security key.

Additionally, TOTP exists to prevent attackers that gained knowledge of the static credentials, e.g., by luring a user to a phishing website, using a keylogger or a user reusing credentials for multiple services, to use them to log into services as the user. In this scenario, TOTP will prevent the attacker from gaining access as they cannot have obtained the TOTP secret need to generate valid TOTP codes. Attackers trying to gain access to a TOTP secured account therefore would need to perform active attacks at the same time the user is, e.g., entering their credentials on a phishing website.

TOTP in general does **not** help in cases where an attacker gains access to a service's credentials database. In this case, the attacker has both the static credentials as well as the TOTP secret. However, even when a user reuses their static credentials, the TOTP secrets are unique per user and service and therefore an attacker cannot gain access to other services the user also has an account with TOTP protection for.

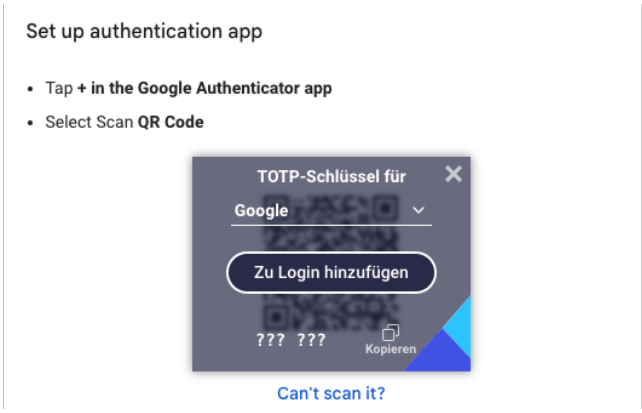

Figure 17: heylogin implements the TOTP setup by detecting QR Codes using the browser extension

In heylogin, you don't need to scan a QR code with your phone's camera to set up TOTP, the browser extension detects the TOTP setup automatically.

When you visit a website that displays a QR code for setting up 2-factor authentication (like in the screenshot), the heylogin browser extension recognizes the QR code directly on the web page. It reads the TOTP secret embedded in the QR code by parsing the image using standard TOTP URI format (e.g. `otpauth://...`). This happens entirely in the browser, no image is sent to a server and no camera is needed.

Once detected, heylogin shows a prompt (like the one in your screenshot), allowing you to directly attach the TOTP secret to the corresponding login entry. This means your 2FA setup is fully integrated with the same 1-click login experience.

5 Threat model

The goal of this threat model is to clearly outline which attacker capabilities, scenarios, and organizational assumptions are considered in the security design of heylogin. By doing so, we make explicit what heylogin protects against, what is considered out of scope, and which additional security measures apply even in unlikely or unfavorable circumstances.

This model covers a broad spectrum of threats, from large scale remote attacks such as infrastructure breaches and MFA fatigue, to targeted scenarios like device theft or the use of hardware keyloggers. Each subsection describes the attacker's capabilities, the possible impact, and how heylogin mitigates or prevents the attack.

A key difference to traditional password managers and MFA systems is that heylogin eliminates the scalable remote attack surface. Instead, attackers are forced into scenarios requiring physical access, which are significantly less practical and non scalable.

5.1 Assumptions

In everyday use, heylogin is typically set up with a phone and one or more paired devices (such as browsers on laptops). Unlock operations require the security chip of the phone and a second factor such as fingerprint, face unlock, or PIN.

Like with any security architecture, the overall protection level depends not only on cryptographic design but also on operational practices. Our threat model makes the following organizational assumption:

- Paired devices (e.g. laptops with a paired browser) are locked with a screen lock when left unattended.

Users can take additional measures to further protect their heylogin access:

- Don't wait for the nightly lock of all paired devices. Instead, configure a shorter lock delay in heylogin or lock a paired device in the heylogin app when stepping away from it.

Scenarios where these assumptions are not met, for example, an unattended, paired and unlocked laptop without screen lock, are considered out of scope for this threat model. However, heylogin still employs additional security measures that limit the impact in such cases. These are described in the sections *Protection in case of a stolen & unlocked phone* and *Additional security measures in case of an unattended paired device*.

5.2 Protection in case of a stolen phone

After stealing a phone, the attacker needs to guess the correct PIN to unlock the security chip. If a biometric factor is enabled, all devices (iPhones, iPads, Android devices, biometric FIDO2 security keys) fall back to PIN-base unlock by design. Thus, most attackers will need to guess the correct PIN since biometric authentication is sensitive to incorrect attempts. In the following, we will provide insights how the brute-force protection on iPhones, iPads, Android devices and FIDO2 security keys work.

5.2.1 Brute-force protection on iPhone and iPad

On Apple devices, the Secure Enclave is responsible for handling user authentication via PIN. The Secure Enclave also implements escalating timeouts on authentication failure to discourage brute-force attacks[6]. Furthermore, Apple allows users to configure an automatic device wipe that will delete all data on the device after 10 incorrect attempts.

Incorrect attempts	1 to 3	4	5	6	7	8	9
Delay in minutes until next try	-	1	5	15	60	180	480

Incorrect attempts	10
Action	Device is disabled or wiped, depending on configuration

All iPhones and iPads supported by heylogin feature a Secure Enclave. Enabling a PIN also enables Data Protection on the device. Data Protection fully encrypts the whole device with a hierarchy of different keys ensuring no unauthorized access to data is possible [7].

heylogin uses Apple's iCloud Keychain to store the user's *platform backup authenticator* securely in Apple's cloud storage. Apple guarantees, that only the user themself can access the data stored in iCloud Keychain[8].

[6] Apple, Passcodes and passwords, 2021, https://support.apple.com/en-gb/guide/security/sec20230a10d/web
[7] Apple, Platform Security, May 2024, https://help.apple.com/pdf/security/en_US/apple-platform-security-guide.pdf, pp. 89ff
[8] Apple, Platform Security, May 2024, https://help.apple.com/pdf/security/en_US/apple-platform-security-guide.pdf, pp. 158-161

5.2.2 Brute-force protection on Android

On Android devices, the Gatekeeper[9] component is responsible for verifying authentication requests made by users via PIN. Due to Androids open nature, Gatekeeper implementations may vary by device vendor and could leverage a TPM, a Trusted Execution Environment (TEE), or both. Request throttling is implemented by Gatekeeper by handling a request timeout value provided by the underlying hardware implementation. The exact timeout values vary by device vendor so we cannot give exact numbers on how many wrong authentications are possible before timeouts kick in. Unfortunately, no concrete information is published by neither Google how their implementation in Pixel devices utilizes the Titan M security chip[10], nor by Samsung regarding their Knox implementation[11]. However, Google has published guidelines and requirements that implementations have to follow[12].

heylogin uses Android's KeyStore[13] and its APIs to securely store the users authenticator seed (see next section). We therefore rely on Android to store this seed in a way other apps cannot access it, which Android guarantees us it does. The exact implementation of the Keystore is device vendor specific, e.g., on Google Pixel devices a dedicated hardware security module is used to store the key material. Other Keystore implementations might leverage the TEE of the SoC instead.

heylogin uses the appropriate Android APIs to make the *platform backup authenticator* available to whatever backup system the phone uses. Typically, this is Google's Cloud Backup service which uses end-to-end encryption with a key derived from the users unlock passcode, PIN or pattern[14]. heylogin cannot make any guarantees regarding the security of other backup solutions on Android.

5.2.3 Brute-force protection with FIDO2 security keys

Independent of our efforts of simplifying website authentication, the FIDO Alliance has been creating the FIDO2 standard which standardizes hardware security key implementations that

[9] Android Source, Gatekeeper, 2024, https://source.android.com/docs/security/features/authentication/gatekeeper

[10] Android, Android Enterprise Security Paper, 2021, https://source.android.com/docs/security/overview/reports/Google_Android_Enterprise_Security_Whitepaper_2020.pdfedd

[11] Samsung, Knox Vault, 2024, https://docs.samsungknox.com/admin/fundamentals/whitepaper/samsung-knox-for-android/core-platform-security/knox-vault/

[12] Android Source, Biometric Sensors, 2024, https://source.android.com/docs/compatibility/14/android-14-cdd#7310_biometric_sensors

[13] Android Source, Android Keystore system, 2024, https://developer.android.com/privacy-and-security/keystore

[14] Troy Kensinger, Google and Android have your back by protecting your backups, 2018, https://security.googleblog.com/2018/10/google-and-android-have-your-back-by.html

allow users to use a hardware security key (commonly referred to as a YubiKey, a specific hardware security key made by Yubico) as a second factor during login and, in newer versions of the standard, even as their only means of authentication.

Hardware security keys have their own key material that never leaves the device and a narrow interface that allows them to perform cryptographic operations on given inputs. In their most basic form, simply possessing the hardware security key is enough to use it. However, this is not secure in case of theft. Therefore, these security keys also require the user to set up a PIN that needs to be provided when cryptographic operations should be performed. Some security keys additionally offer a biometric unlock option. In those cases, a fallback to PIN is typically done after a set amount of incorrect scans, the exact amount is vendor dependent, e.g. 3 attempts[15] or 5 attempts[16].

Similarly to smartphones, FIDO2 certified security keys also implement a limit of 8 incorrect PIN attempts[17]. After the last incorrect attempt, the security key needs to be reset, deleting all stored key material[18].

Incorrect attempts	1 to 7	8
Action	-	Security key is disabled and has to be reset to factory defaults to work again

5.3 Protection in case of a stolen & unlocked phone

Even in the unlikely case that someone gains access to an unlocked phone, heylogin ensures that possession of the phone alone is never enough to compromise accounts. Sensitive operations are always protected by an extra security layer of local re-authentication.

heylogin does not rely solely on the phone's global state of its screen unlock. Instead, it requires a second explicit unlock for sensitive operations. Specifically, the user must authenticate again with fingerprint, face unlock, or PIN before allowing any of the following operations:

[15] Yubico, Yubikey Bio, 2024, https://docs.yubico.com/hardware/yubikey/yk-tech-manual/bio-specifics.html#yubikey-bio-and-fido2

[16] Onespan, DIGIPASS FX1 BIO, 2024, https://www.onespan.com/sites/default/files/2023-11/digipass-fx1-bio-user-manual.pdf §1.3

[17] FIDO Alliance, Client to Authenticator Protocol (CTAP), 2019, https://fidoalliance.org/specs/fido-v2.0-ps-20190130/fido-client-to-authenticator-protocol-v2.0-ps-20190130.html#retries

[18] FIDO Alliance, Client to Authenticator Protocol (CTAP), 2019, https://fidoalliance.org/specs/fido-v2.0-ps-20190130/fido-client-to-authenticator-protocol-v2.0-ps-20190130.html#authenticatorReset

- Unlocking a paired device
- Pairing another device
- Opening the login list within the heylogin app
- Autofilling logins in other apps or browsers via the system-level Autofill API

The first time a sensitive operation is performed, the user must authenticate. The app then remains unlocked as long as it stays in the foreground. If the phone is locked, the user switches to another app, or the app has been in the background for a while, re-authentication is required again.

In addition, some operations always require explicit re-authentication, no matter the app state:

- Show backup code
- Delete account

Additionally, heylogin refuses to run on devices without a system screen lock (fingerprint, face unlock, or PIN) enabled.

5.4 Additional security measures in case of an unattended paired device

If a paired device (for example, a laptop with a paired browser) is left unattended without a screen lock enabled, and heylogin is still unlocked for this device because the lock delay has not yet expired, an attacker with physical access could potentially use it.

The primary mitigation is the lock mechanism described in *Locking heylogin again*: users should lock their paired device when leaving it unattended or configure a shorter lock delay in heylogin.

As an additional security measure, heylogin requires explicit confirmation on the phone for highly sensitive operations in the heylogin webapp. These operations include:

- Export of stored logins
- Show or generate backup code
- Adding additional login devices (FIDO2 security keys, Windows Hello, or Touch ID)
- Delete an organization

This design ensures that an attacker with temporary access to an unlocked paired device cannot immediately perform the most sensitive operations without also having access to the phone and passing biometric or PIN authentication.

5.5 Protection in case of an infrastructure breach

heylogin provides extensive hardware protection even in case of an infrastructure breach.

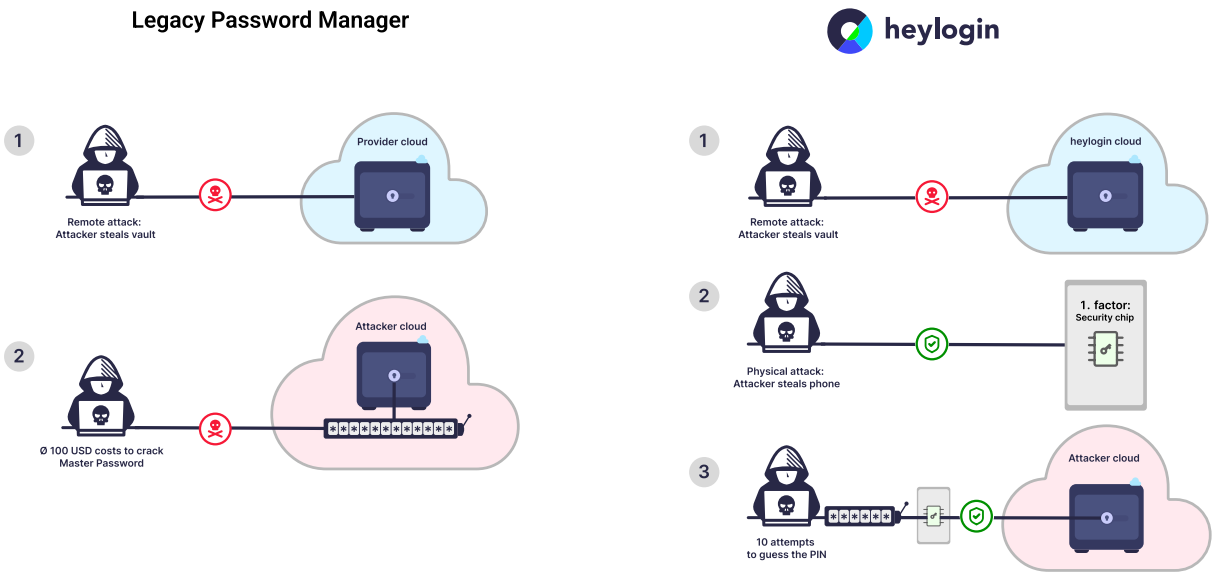

Figure 18: If vaults are stolen, the attacker has an unlimited number of attempts to crack the master password. In comparison, with heylogin the attacker has to physically steal the security chip and only has a limited number of attempts to guess the correct PIN. With iOS, for example, after 10 incorrect attempts the device blocks completely.

If someone breaks into the servers of a legacy Password Manager provider (**step 1**) and steals the encrypted vaults, the attacker has an unlimited number of attempts to crack the Master Password (**step 2**). This has been discussed extensively in previous sections.

In contrast, if someone breaks into heylogin's servers and steals our customers' encrypted vaults (**step 1**), the attacker also needs to **physically** steal the customer phones (**step 2**). This is already highly unlikely since most real world attacks are carried out remotely by attackers from foreign countries. Brute-forcing the seed directly that has been generated inside the security chip is not possible since it contains high amounts of entropy.

An attacker physically stealing the phone (**step 3**) is limited by the number of PIN retries as discussed in *Protection in case of a stolen phone*.

Even if it's a highly targeted attack and the attacker successfully steals a customer phone, it's virtually impossible to steal a large amount of customer phones. Thus, while brute-forcing Master Passwords scales, stealing phones does not scale.

5.6 Post-Compromise Security

In cryptographic systems, two related but distinct security properties are relevant in the context of compromises: Forward Secrecy and Post-Compromise Security. While both limit the impact of key compromise, they do so at different points in time.

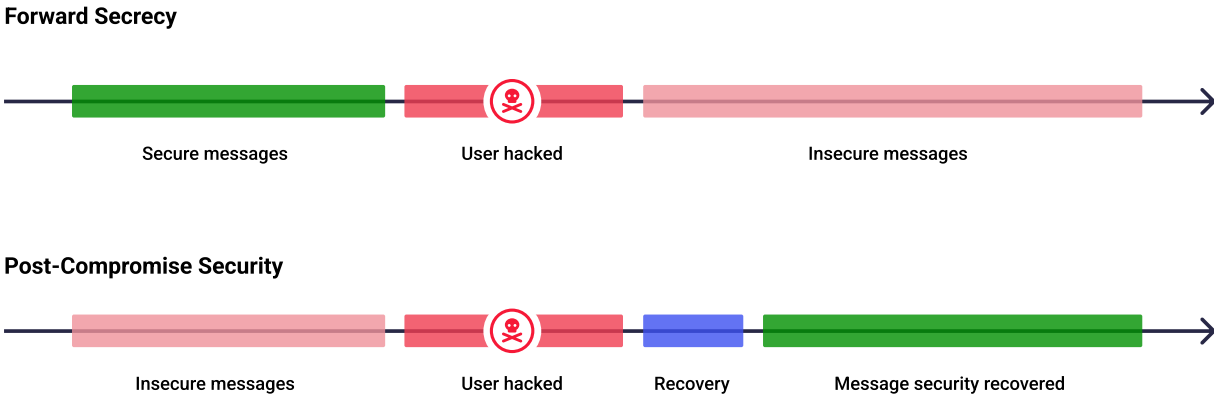

Figure 19: Difference between Forward Secrecy and Post-Compromise Security in secure messaging protocols

As shown in the figure, **Forward Secrecy** protects data created **before** a compromise. Even if key material is stolen, previously encrypted data remains confidential. This is typically achieved using ephemeral session keys that are deleted after use. **Post-Compromise Security** instead focuses on recovery **after** a compromise. While data exposed during the compromise window may be lost, confidentiality can be restored for future encrypted data once a recovery event occurs, such as key rotation. Secure messaging protocols, such as the Signal protocol, implement cryptographic mechanisms to ensure both Forward Secrecy and Post-Compromise-Security to protect message before and after a compromise.

Password managers operate under a different usage model than messaging protocols. Vault data must be stored persistently and decrypted repeatedly over time. As a result, Forward Secrecy is architectually not achievable for password managers, as encrypted vaults must remain decryptable by design. Legacy password managers also typically provide no Post-Compromise Security, with few exceptions such as KeePass where the vault is re-encrypted when the Master-Password is changed. In most cases, the same long term key material or Master-Password-derived key protects the vault over its entire lifetime.

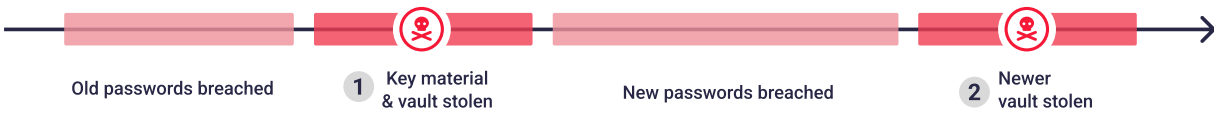

Figure 20: Without Post-Compromise Security new passwords generated after a device compromise are breached again when a newer vault is stolen.

As illustrated in the figure, this leads to a fundamental problem. In the **first compromise event (1)**, an attacker gains access to the vault and its key material, for example through a compromised client device. This material is typically derived directly from the master password, making the vault decryptable once both the vault and the master password are obtained. This allows the attacker to decrypt all **old passwords** until the compromise event.

In the **second compromise event (2)**, a newer version of the vault is stolen, for example through an infrastructure breach. Because the vault is still encrypted with the same long term key material, the attacker can decrypt it as well, compromising even the **new passwords**. This illustrates that even if the user detects the first compromise and changes all affected website passwords and the master password, those new passwords are compromised again when a newer version of the vault is stolen.

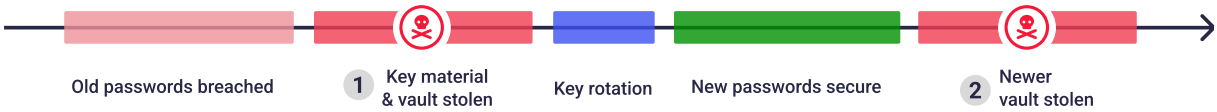

Figure 21: With Post-Compromise Security new passwords generated after a device compromise stay secure even in case a newer vault is stolen.

In contrast, heylogin provides Post-Compromise Security through automated key rotation. In the **first compromise event (1)**, an attacker gains access to the vault and its key material, for example through a compromised client device. This allows the attacker to decrypt all **old passwords** until the compromise event. Once the compromise is detected, affected devices can be disconnected by organization admins, or users can initiate recovery mechanisms to remove compromised devices. After this, key rotation is performed and all subsequent vault updates are encrypted with new key material.

In the **second compromise event (2)**, a newer version of the vault is stolen, for example through an infrastructure breach. Unlike with legacy password managers, the attacker can no longer decrypt the updated vault contents, as the compromised keys from the first event are no longer valid. As a result, **new passwords** that are changed or newly added after the compromise remain confidential.

Details on when and how key rotation is performed and how vaults are re-encrypted are described in *Key rotation*.

5.7 Attacks via malicious QR codes

As described in *Pairing app with browser extension*, heylogin uses QR codes to establish an out of band authenticated key exchange when pairing devices. Further cryptographic details are described in *Pairing another device*. While this design prevents remote and fully automated attacks, QR codes introduce a limited attack surface if an attacker succeeds in presenting a malicious QR code to a user. These attacks are sometimes called QRLJacking, Quishing or GhostPairing.

- If a QR code is scanned directly using the operating system's camera application, the heylogin app is opened automatically. Scanning a QR code alone is not sufficient to pair a new device. heylogin always requires an explicit additional confirmation by the user that the scanned device should be paired.
- A phishing based attack may redirect the user to a malicious website imitating heylogin.app and displaying an attacker controlled QR code. Such attacks can be combined with Browser in the Middle techniques, where the attacker transparently proxies all browser interactions between the victim and the legitimate service. The victim interacts with what appears to be the real service, while all actions, including pairing confirmations, are relayed and executed under the attacker's control. Protection against phishing based QR code attacks is limited when pairing new devices via smartphones. FIDO2 security keys provide phishing resistant authentication, as the browser cryptographically binds them to the legitimate heylogin.app origin. Organizations with elevated security requirements should therefore rely on FIDO2 security keys.

5.8 MFA fatigue attacks

MFA fatigue attacks (also known as push bombing or prompt spamming) are a widespread social engineering strategy targeting push-based MFA systems used by providers such as Okta, Google, and Microsoft. In these attacks, adversaries use compromised passwords, often from data breaches, to repeatedly trigger MFA prompts until the user, fatigued or confused, approves one[19].

[19] Cybersecurity and Infrastructure Security Agency (CISA). *Implementing Phishing-Resistant MFA*. October 2022, https://www.cisa.gov/sites/default/files/publications/fact-sheet-implementing-phishing-resistant-mfa-508c.pdf

Real-world incidents have shown the effectiveness of this technique. In 2022, the hacking group Lapsus$ used MFA fatigue attacks to compromise both Okta and Uber. At Okta, attackers targeted a contractor, bombarding them with MFA prompts until access was granted, potentially impacting around 2.5% of Okta's customers. At Uber, a contractor was similarly overwhelmed with MFA prompts and ultimately tricked into approving one, giving the attackers access to internal systems[20][21].

With heylogin, this attack vector fundamentally breaks down:

- All MFA fatigue attacks start with compromised passwords from data breaches, which enable attackers to send repeated MFA prompts.
- In heylogin, however, there is nothing usable to steal remotely or obtain from a data breach. MFA prompts can only be sent to already paired devices.
- To proceed, an attacker would need to:
  - **Option A:** Coerce the user into pairing a new device via QR code, as discussed in the previous section.
  - **Option B:** Physically steal the paired device, such as the user's laptop, and bypass its screen lock. Then open heylogin.app and trigger the unlock notification. This scenario is technically possible but already requires physical access, which typically not assumed for an MFA fatigue attack. In addition, users can remove paired devices that have been stolen using the heylogin app.

As a result, heylogin eliminates the remote, scalable nature of MFA fatigue attacks. The threat model shifts from large-scale remote exploitation to isolated physical theft, a completely different and far less practical class of risk.

5.9 Protection against hardware keyloggers

Hardware keyloggers are small devices that can be secretly placed between a keyboard and a computer to capture everything typed, including passwords. Such attacks are a classic threat against systems these type of systems.

With heylogin, this attack vector is ineffective for two reasons:

- **No password entry:** Users do not type passwords at all. Logins are managed by the heylogin app and transmitted securely, so there is nothing for a keylogger to capture during normal use.

[20] Security Risk Advisors (SRA). OKTA Breach Background & Impact Recommendations – TIGR Team. March 2022, https://sra.io/wp-content/uploads/2022/03/SRA-TIGR-RFI-OKTA-Breach.pdf

[21] TechCrunch. How to fix another Uber breach. September 19, 2022, https://techcrunch.com/2022/09/19/how-to-fix-another-uber-breach/

- **Short-lived secrets:** Even if a hardware keylogger records the characters of a start code or PIN during the pairing process, these codes are strictly single-use and expire immediately. They cannot be reused by an attacker.
- **Ephemeral memory use:** Logins are only kept in device memory for the duration of the authentication process and are not persistently stored where a keylogger could repeatedly access them.

As a result, keystrokes intercepted by a hardware keylogger do not provide attackers with reusable secrets. heylogin is therefore resistant to one of the oldest and most widely known hardware-based credential theft techniques.

5.10 Security of passwords in memory

If passwords remain in memory for too long, malware or forensic tools can extract them and compromise accounts. A study by secuvera showed that many traditional password managers leave cleartext passwords in memory for extended periods, making them vulnerable to such attacks[22]. These are attack scenarios where no solution can fully protect against an adversary that has already installed malware on the system and gained access to memory.

With heylogin, the time that logins remain in memory during the website authentication step is minimized. This reduces the attack surface for memory scraping techniques. In response to secuvera's blog post, heylogin was the only password manager to address this attack scenario and adapt its design accordingly.

5.11 Additional threat model limitations

Further limitations, assumptions, and out-of-scope scenarios are discussed in *Appendix A: Threat Model Limitations*.

6 Cryptographic architecture

The remaining sections will go into the details of heylogin's cryptographic architecture. It details the main algorithms, keys and security protocols. It is intended for security engineers and cryptographers.

[22] secuvera GmbH. Studie: Klartextpasswörter in Passwortspeichern. August 2024, https://www.secuvera.de/blog/studie-klartextpassworter-in-passwortspeichern/

6.1 Cryptographic algorithms and key notation

This document presents the architecture and algorithms used by heylogin to store data. heylogin only uses state-of-the-art cryptography algorithms. On all platforms we utilize libraries that implement the NaCl interface and therefore use Curve25519[23] for asymmetric cryptography and the XSalsa20[24] stream cipher with Poly1305[25] authentication for symmetric cryptography. When asymmetric keys need to be derived, we use NaCl's implementation of SHA-512 as a key derivation function and truncate the ouput to 32 byte. At the time of writing, the library used on all platforms is TweetNaCl.js[26]. On mobile, sometimes native code is needed in which case we also use libsodium. For Argon2[27] we use different wrappers of the Argon2 reference implementation depending on the platform.

Within heylogin, keys are named to indicate whether they are allowed to be saved on the device or not. In this document we will use the term *storable* to indicate that a key is allowed to be stored on the device and the term *high security* to indicate that the key may only exist in volatile memory and must not be persisted. Such keys must be derived or acquired before using them. Asymmetric key pairs consist of two keys, the secret and public key, which will use the following notation: $usage_{type}^{prefix}$ where *prefix* is either *s (storable)* or *hs (high security)*, *type* is either *sec (secret)* or *pub (public)* and *usage* is a unique name indicating what this key is used for. Symmetric keys use a similar notation, just without the key type: $usage^{prefix}$.

6.2 Architecture overview

The architecture of heylogin consists of two parts, the server side which stores encrypted customer data and the client applications talking to the server side. There are two kinds of clients available, the Authenticator, which is the mobile app, and the App, which are the web app and the extension.

The following Figure shows how the parts communicate with each other. All client applications talk to the server side using TLS 1.3, depicted by solid arrows. The web app and extension can exchange data on the local machine directly inside the browser depicted by a dashed arrow. The authenticator cannot talk directly to the web app or extension and therefore the server side acts as a proxy. All messages between the apps and the authenticator are always end-to-end

[23] Internet Research Task Force (IRTF), Elliptic Curves for Security, 2016, https://datatracker.ietf.org/doc/html/rfc7748

[24] Daniel J. Bernstein, Extending the Salsa20 nonce, 2011, https://cr.yp.to/snuffle/xsalsa-20110204.pdf

[25] Internet Research Task Force (IRTF), ChaCha20 and Poly1305 for IETF Protocols, 2015, https://datatracker.ietf.org/doc/html/rfc7539

[26] Dmitry Chestnykh, TweetNaCl.js, https://tweetnacl.js.org

[27] Alex Biryukov, Daniel Dinu, and Dmitry Khovratovich, Argon2: the memory-hard function for password hashing and otherapplications, 2017, https://www.cryptolux.org/images/0/0d/Argon2.pdf

encrypted and sent over the TLS connections, the virtual communication channels are depicted as dotted arrows.

On the server side, all user data is saved encrypted with keys that are only available on the user's devices. The only accessible data is the user's email address and structural metadata, such as which data belongs to which user. The apps never send unencrypted user data to the server-side. The heylogin server-side merely act as data storage and communication proxy.

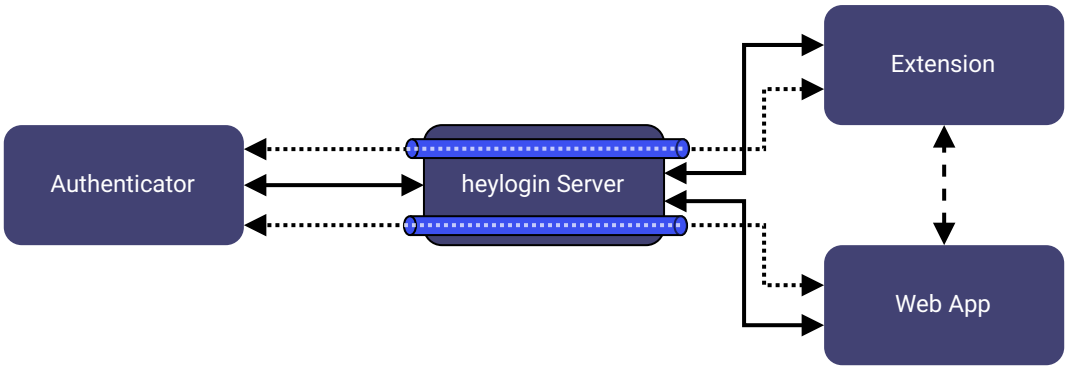

Figure 22: Overview of the heylogin architecture. Solid arrows indicate TLS-encrypted connections over the internet, dotted arrows indicate E2E-encrypted connections inside the TLS-encrypted connections and dashed arrows indicate a local, unencrypted connection on the same device.

6.3 Authenticators & profiles

heylogin uses multiple layers of indirection and key material to facilitate encryption of data. The main layers are *Authenticators*, *Profiles* and *Vaults*. A user has one or multiple authenticators which in turn enable cryptographic access to one multiple profiles. Profiles then enable cryptographic access to vaults which store the actual encrypted user data. In organization and team scenraios, users only interact with profiles of other users, never authenticators, to simplify the cryptographic operations needed when keys need to be rotated. The exact details of each layer and their interactions are shown in the following sections.

6.3.1 The smartphone as a Push Authenticator

One of the core concepts of heylogin is the *Authenticator*. With heylogin, every user has at least one authenticator: their mobile phone. Modern mobile phones and their operating systems allow applications to store secrets and sensitive data securely by offering support for cryptographic operations with key material only available to the hardware. This hardware is known as

a *secure element* and can encrypt and decrypt user data on request. The heylogin app utilizes the secure element to create an authenticator in the following way: First, a random 32-byte seed is generated on the mobile phone. This seed is then given to the secure element to be encrypted and the encrypted form is saved on the device. The seed is then used to derive the following five key pairs mixing in fixed info to scope the keys and using .

$$\begin{aligned}
\left(login^{hs}_{\text{sec}}, login^{hs}_{pub}\right) \\
\left(id^{hs}_{\text{sec}}, id^{hs}_{pub}\right) \\
\left(profileSeedEnc^{hs}_{\text{sec}}, profileSeedEnc^{hs}_{pub}\right) \\
\left(profileSeedEnc^{s}_{\text{sec}}, profileSeedEnc^{s}_{pub}\right) \\
\left(sig^{s}_{\text{sec}}, sig^{s}_{pub}\right)
\end{aligned}$$

All public keys are saved on the server side. To derive all key pairs except the login key pair, a unique salt is also randomly generated and saved on the server side. The login key pair is used for authenticating this authenticator against the server. The id key pair represents the identity of this authenticator. Both $profileSeedEnc$ key pairs are used to encrypt two types of profile seeds.

The $profileSeedEnc^{hs}_{pub}, profileSeedEnc^{s}_{pub}, sig^{s}_{pub}$ public keys are signed using id^{hs}_{sec} and the signatures are saved on the server in addition to the public keys. This prevents the server from changing any of the public keys without also changing id^{hs}_{pub} and therefore changing the identity of the authenticator. The user now has a so called *Push Authenticator*. It is called a push authenticator as it can receive push notifications that prompt the user to unlock secrets for a client.

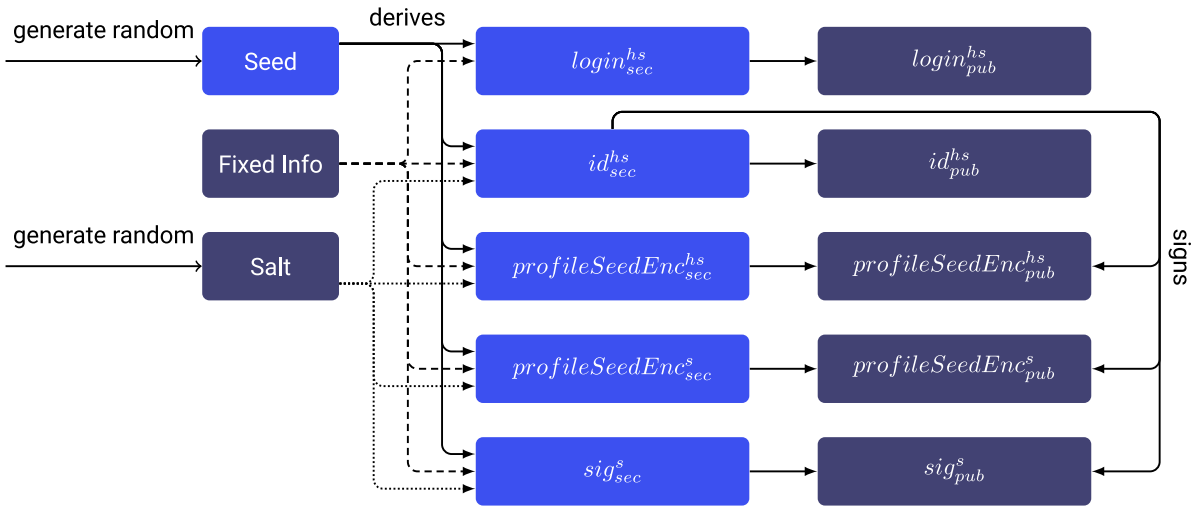

Figure 23: Key generation for authenticators

6.3.2 FIDO2 devices as additional authenticators

heylogin supports adding FIDO2 authenticators as a additional login devices besides the smartphone for unlocking sessions. This could be roaming authenticators, such as security keys or smartcards, or platform authenticators, such as Windows Hello or Apple TouchID[28].

Using FIDO2 devices as true standalone authenticators requires browser support of the WebAuthn PRF extension. This extension exposes the FIDO2 hmac-secret extension to store or generate key material on FIDO2 devices. Since not all browser and operating system combinations support the WebAuthn PRF extension, we work around this restriction by allowing FIDO2 authenticators to use the authenticator seed of the mobile client in the following way: First, a random challenge is generated on the server side and send to the FIDO2 authenticator. The FIDO2 authenticator signs the challenge and returns it to the server side together with the devices public key and a credential which the server side saves. This pairs the FIDO2 authenticator to the user. The server side adds a flag to the user, that the encrypted seed shall not be deleted when unlocks expire or if sessions are locked, see *Locking and unlocking a device*.

When an unlock request is made, the client side requests a challenge and all registered credentials from the server side which is forwarded to the FIDO2 authenticator. The FIDO2 authenticator checks the provided credentials to identify if it in the list of expected devices. If so, it signs the challenge and returns it. The server side verifies the challenge with the saved public keys. If

[28] W3C Recommendation, Web Authentication: 6.2. Authenticator Taxonomy, 2021, https://www.w3.org/TR/webauthn-2/#sctn-authenticator-taxonomy

the signature matches, the saved encrypted seed of the mobile session is returned to the client which can then derive the needed keys to complete the unlock.

heylogin enforces that the FIDO2 authenticators authenticates the user, e.g., via a PIN or a biometric unlock, by checking the signed returned result of the FIDO2 authenticator. If an error occurred, the device does not support authentication or authentication was skipped or failed, heylogin will abort the unlock request.

FIDO2 platform authenticators, such as Windows Hello and Apple TouchID, are listed distinctly in heylogin as they are internal to one specific physical computer. During the paring process therefore, they are not bound to the whole user but to the session that initiated the pairing and can only be used to unlock that specific session.

If the PRF extension is available, heylogin can use the FIDO2 authenticator to store or generate key material which allows the use of FIDO2 authenticators as true standalone authenticators. In this case, we can "upgrade" the security key by letting the security key create an authenticator seed for us that we can then use to create new authenticator key pairs and locks for the profiles. This changes the authenticator to a *Webauthn Authenticator*. This authenticator has no session limit as it has the authenticator material available and can connect new sessions if desired. The authenticator now behaves similar to a *Push Authenticator*, e.g., the encrypted authenticator seed is deleted when a session is locked.

6.3.3 Platform Backup Authenticator

To prevent loosing access to heylogin should the push authenticator be unavailable, e.g. because the mobile phone is broken, a *Platform Backup Authenticator* is generated in the same way as the push authenticator. This platform backup authenticator seed is stored using the backup API of Android[29] or iOS[30]. We ensure that this seed is end-to-end encrypted by limiting this functionality to devices that support encrypted backups.

On Android, this functionality of heylogin is only used on devices with Android 9 or later. In addition, this backup mechanism is only enabled for backups that are transported using a device-to-device transfer and for client-side encrypted backups. We ensure that heylogin's backup functionality is restricted to these two scenarios by only allowing backups that either have the `FLAG_DEVICE_TO_DEVICE_TRANSFER` or `FLAG_CLIENT_SIDE_ENCRYPTION_ENABLED` enabled during BackupAgent.onBackup[31].

[29] Troy Kensinger, Google and Android have your back by protecting your backups, 2018, https://security.googleblog.com/2018/10/google-and-android-have-your-back-by.html

[30] Apple Documentation, iCloud data security overview, 2023, https://support.apple.com/en-us/HT202303

[31] Android Developers, BackupAgent.onBackup API, 2023, https://developer.android.com/reference/android/app/backup/BackupAgent#onBackup(android.os.ParcelFileDescriptor,%20android.app.backup.BackupDataOutput

Device-to-device transfer happens when migrating from one Android device to another Android device by connecting them over USB (or WiFi). It uses no third party or intermediate storage.

Client-side encrypted backups work by generating a random key at the client, which is used to encrypt the backed-up application data. This decryption key is then encrypted using the user's lockscreen PIN, pattern, or passcode, which is not known by Google. This protected key material is encrypted to a Titan security chip in Google's datacenter. The Titan chip is configured to only release the backup decryption key when presented with a correct claim derived from the user's passcode. Because the Titan chip must authorize every access to the decryption key, it can permanently block access after too many incorrect attempts at guessing the user's passcode, thus mitigating brute-force attacks. The limited number of incorrect attempts is strictly enforced by a custom Titan firmware that cannot be updated without erasing the contents of the chip. By design, this means that no one (including Google) can access a user's backed-up application data without specifically knowing their passcode.

On iOS, the platform backup authenticator seed is stored using the iCloud Keychain API. When setting `kSecAttrAccessibleWhenUnlocked`, iOS ensures us that this backup is end-to-end encrypted[32]. This means that the seed is only available when the keychain is unlocked and enforces that the seed is encrypted when transmitting it to a new device.

6.3.4 Backup Code Authenticator

Another type of recovery method exists via a backup code displayed within the app. This code represents an optional *Backup Code Authenticator*, which is created when the code is first retrieved.

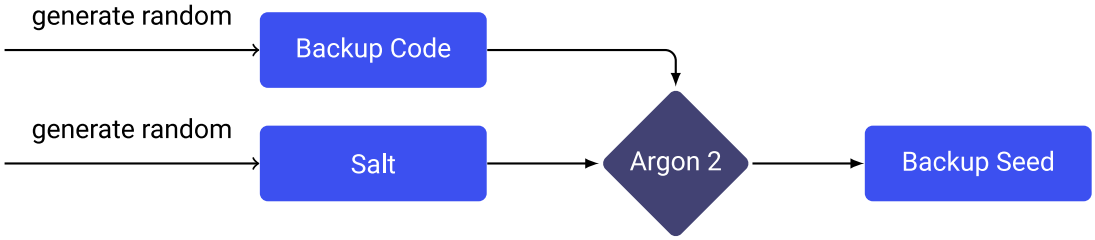

Figure 24: Generation of backup seed based on backup code and salt.

The backup code consists of 24 random digits, each in the range 0-9, formatted into 6 blocks of 4 digits (e.g., `9132-8293-5691-3554-2127-1233`). This allows for 10^{24} possible combinations,

,%20android.os.ParcelFileDescriptor)

[32] Apple Developer Documentation, kSecAttrAccessibleWhenUnlocked method, 2023, https://developer.apple.com/documentation/security/ksecattraccessiblewhenunlocked

since each of the 24 digits has 10 possibilities. This corresponds to an entropy of approximately

$$\log_2(10^{24}) \approx 79.7 \text{ bit}.$$

That's significantly stronger than typical human-created passwords (20–40 bits) and aligns with cryptographic best practices.

Alongside the backup code, a 256-bit salt is randomly generated and stored server-side. This salt is retrieved when needed.

To derive the actual 256-bit backup seed used by the *Backup Code Authenticator*, the backup code is processed using the modern key stretching algorithm Argon2[33]. In heylogin, Argon2 is used to derive an encryption key (not for password hashing), so we apply a more demanding configuration than typically recommended by OWASP[34]. Argon2 is configured with 6 iterations, 2 threads, and 48 MiB of memory, causing each attempt to take 50-150 ms on standard hardware. This limits attackers to ~20 guesses per second, even with good hardware, making brute-force attacks practically infeasible.

With, high entropy (≈80 bits), a unique 256-bit server-side salt and strong Argon2 key-stretching, the *Backup Code Authenticator* reaches a level of security, robust even against offline brute-force attempts.

6.3.5 Profiles

When multiple users need to interact with each other, they can use the authenticators of the other users to securely share data with them. Since every user has multiple authenticators, however, this becomes cumbersome as data has to be bound to all authenticators a user possesses and, in case of recovery, data has to be re-encrypted multiple times.

[33] Alex Biryukov, Daniel Dinu, and Dmitry Khovratovich, Argon2: the memory-hard function for password hashing and otherapplications, 2017, https://www.cryptolux.org/images/0/0d/Argon2.pdf

[34] OWASP, Password Storage Cheat Sheet, 2025, https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html

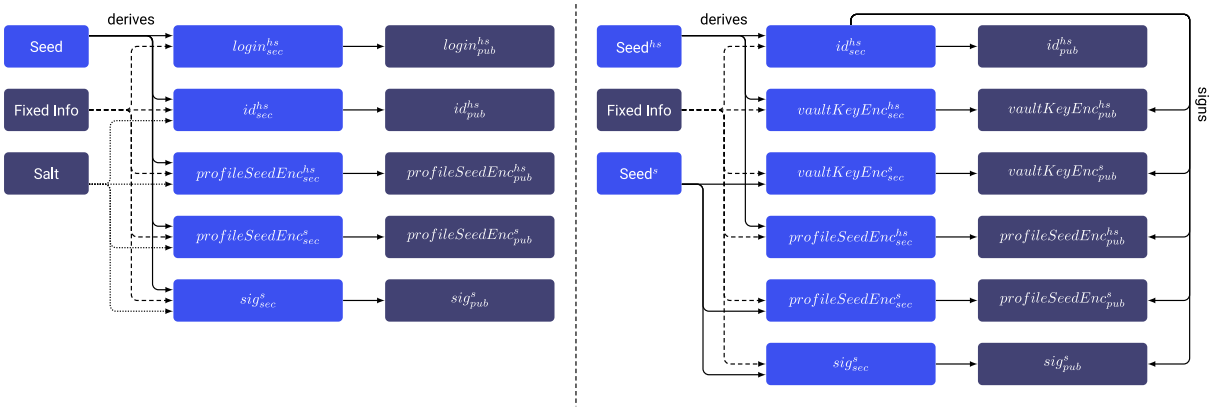

Figure 25: Differences in key material between authenticator and profiles.

We therefore introduce an abstraction layer called *Profiles*. Profiles have similar capabilities as authenticators as they possess similar key types. Profiles do not posses a $(login^{hs}_{sec}, login^{hs}_{pub})$ key pair and therefore cannot be used to log into an account but do have all other key pairs of an authenticator. Furthermore, profile keys are derived from two random seeds and fixed info whereas authenticators are only derived from one seed. This makes it possible for a locked session to access all vault data marked as storable.

Profiles are tied to an authenticator through a *Profile-Authenticator-Lock* which encrypts the profile seeds to the different authenticators a user has. Specfically, the $Seed^{hs}$ is encrypted with the $profileSeedEnc^{hs}_{pub}$ and the $Seed^{s}$ is encrypted with the $profileSeedEnc^{s}_{pub}$ of the authenticator. Notice that the profile also has two $profileSeedEnc$ keypairs. These are used for *Profile-Profile-Locks* in Organisation contexts.

After authenticator creation, the mobile app creates two profiles for the user: the *Inbox Profile* and the *Preferences Profile*. The inbox profile is used by other users when they need to encrypt something only for one specific user. It can be interpreted as the public address of that user. The preferences profile is used to access the *Preferences Vault* which stores metadata such as the names of *Sessions* and the optional Backup Code of the Backup Code Authenticator so that the code can be shown again across multiple devices.

Optionally, users can create a *Private Profile* which accesses the *Private Vault* which stores *Logins* that are private to that user. This is the free for personal use scenario that heylogin offers.

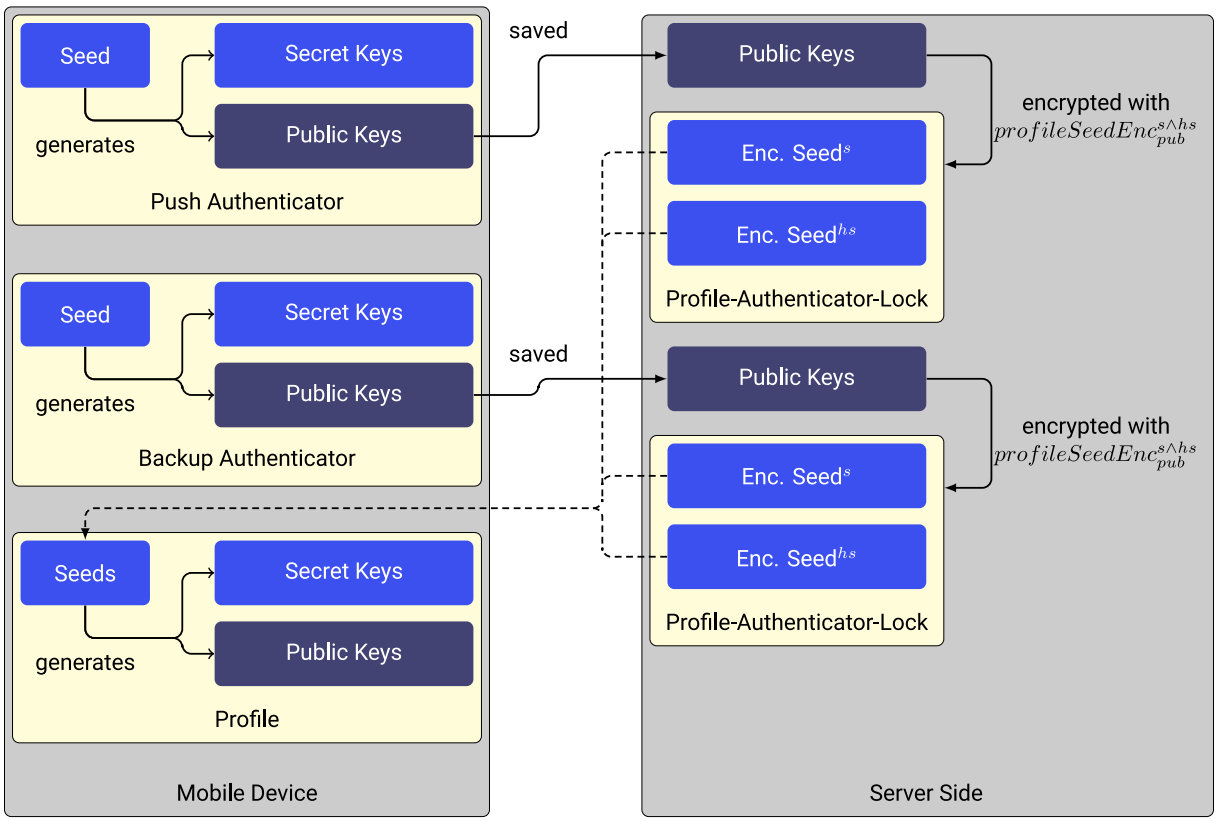

Figure 26: Overview of how *Profile-Authenticator-Locks* are used.

6.4 How logins are secured

6.4.1 Saving and reading logins

The following overview depicts the architecture for saving and reading logins. Logins are saved inside a *Vault* as part of a *Commit*. A login consists of at least one associated website, a username or email address and a password. The website and username can be empty but a password must be set. Additionally, a login can contain custom fields with a freely choosable name and value. Optionally, the value of a custom field can be marked as protected which causes it to be treated as if it were a password. Also optionally, a TOTP secret can be saved which is also treated as if it were a password. Other login types are also possible, for example Credit Cards or WiFi networks.

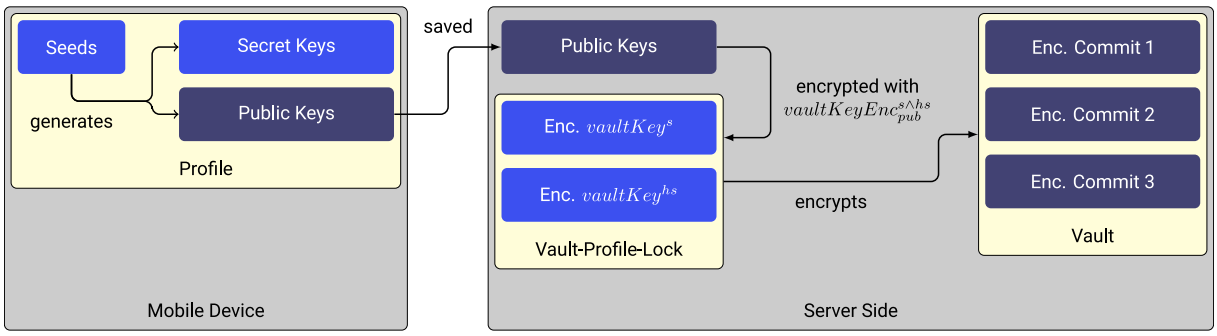

Figure 27: Overview for saving and reading logins. Logins are stored inside commits which are encrypted with the symmetric vault keys. These keys are encrypted with the public keys of a profile to form a Vault-Profile-Lock.

Logins are saved inside *Vaults*. Each vault consist of a series of *Commits* that, when applied in order, represent the current state of the vault. Commits are not cryptographically linked but are ordered by their creation time on the server side. Commits are encrypted with a symmetric key called $vaultKey^s$. Additionally, sensitive parts inside the commit, e.g., the password, TOTP secret and custom fields marked as protected, are additionally encrypted with another symmetric key called $vaultKey^{hs}$.

The symmetric vault keys are saved in *Vault-Profile-Locks* and allow a specific profile to access them. Notice how the profiles are used as an indirection here so that only one profile is used instead of multiple authenticators per user.

The $vaultKey^s$ and $vaultKey^{hs}$ are stored encrypted on the server side inside a *Lock*. The lock binds the $vaultKey^s$ and $vaultKey^{hs}$ to a specific profile. The $vaultKey^s$ is encrypted with $vaultKeyEnc_{pub}^{s}$ and the $vaultKey^{hs}$ is encrypted with $vaultKeyEnc_{pub}^{hs}$ of the respective profile. For each vault, exactly one lock per profile exists that holds the encrypted $vaultKey^s$ and $vaultKey^{hs}$.

6.4.2 Pairing another device

A client can be in one of three states with different key material available to it as shown in the following Figure.

Figure 28: Different states of a client device and which keys are available to it.

After the mobile device has been set up, other devices, such as a browser, here called client, can be paired. Pairing is done by navigating to heylogin.app and then scanning the displayed QR code with the mobile device. The QR code embeds a public key of an ephemeral key pair $(ephemeral_{sec}, ephemeral_{pub})$ which is generated by the client. The SHA256 hash of that public key is send to the server side which proxies the reply of the mobile device to the web app. The content of the QR code is the URL `https://heylogin.app/qr/#` concatenated with $Base64(ephemeral_{pub})$.

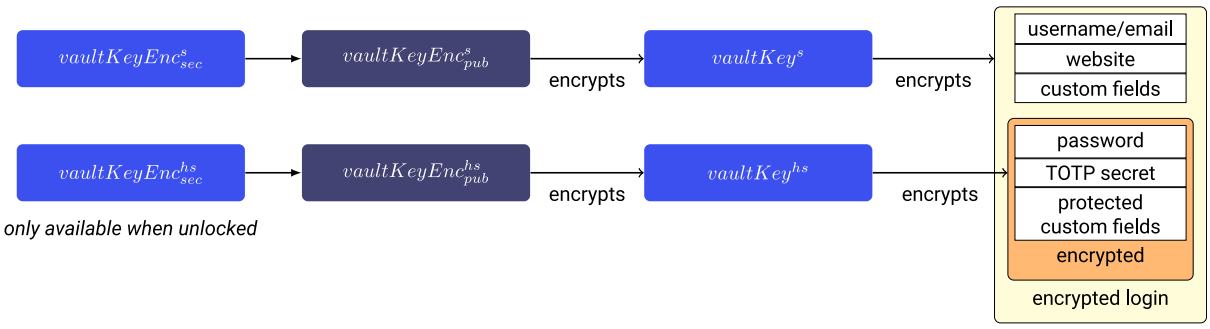

Figure 29: In the paired, but locked state, username, website and other custom fields are available. Passwords, TOTP secrets and other protected custom fields are only available in the unlocked state.

Should the URL be scanned by another QR code scanner app that is not the heylogin mobile app and subsequently be opened in a web browser, then the server will simply issue a redirect to the web app. Furthermore, the heylogin mobile app registers itself as a handler for the `https://heylogin.app/qr/` URL so that it is opened automatically. To prevent a malicious actor from pairing a new session by simply invoking the heylogin app QR code URL handler, the app forces the user to scan the QR code again and ignores the passed URL.

The following process is also depicted in the following Figure. Upon scanning the QR code with the heylogin mobile app, the mobile device encrypts its push authenticator seed to that public key, sends it to the server side which relays it to the client who then decrypts the seed to generate the login key pair. With this key pair, the client authenticates itself against the server side and obtains the salts to generate all other authenticator key pairs. It now generates a session key $(session^{s}_{sec}, session^{s}_{pub})$, signs $session^{s}_{pub}$ with the authenticator identity secret key id^{hs}_{sec} and saves $session^{s}_{pub}$ and the signature inside the personal meta vault. This mechanism creates a *session* on the server side for this client. The authenticator seed is also encrypted with the $session^{s}_{pub}$ and saved alongside the session so that the client can retrieve it again later.

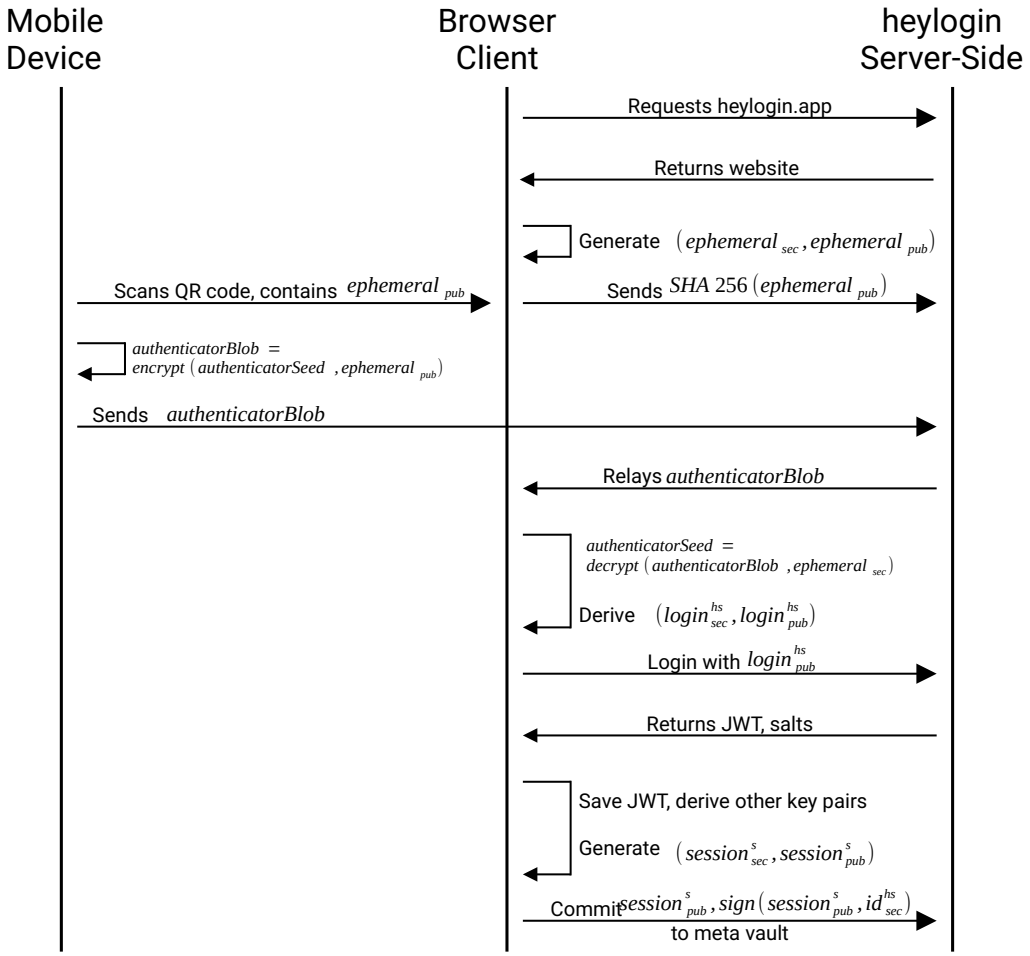

Figure 30: Pairing a new device

Each session also has a JSON Web Token (JWT) generated by the server side containing the session and user id. The JWT authorizes this client to send requests to the server side. This is also true for push authenticators which also have a session. All sessions except the own are displayed in the mobile app and can be deleted if needed. When a session is deleted, the respective session id on the server side is deleted and the client can no longer perform a request to the server side. Furthermore, the deleted session's public key and signature are invalidated due to the now deleted session id. A client that gets its request rejected due to a deleted session will delete all key material it has.

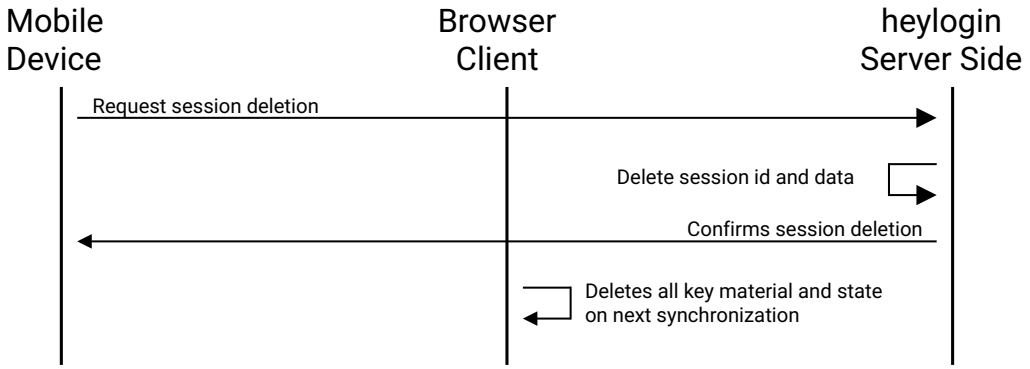

Figure 31: Deleting a paired device

So far, a client has been described as being the heylogin.app web app. The heylogin extension installed inside a browser with a paired web app will communicate with the web app and use the same session credentials and therefore will be identified as the same device as the web app. The extension will also save the session credentials to be able to restore the session for the web app even if the browser cookies and local storage is deleted.

6.4.3 Locking and unlocking a device

A device being locked or unlocked is an implicit state. The presence of an encrypted authenticator seed inside a session means that the session is unlocked. If the encrypted seed is not present, the session is locked. The mobile device can add and remove the encrypted seed to and from sessions at will. These are the toggles inside the mobile app beside each session. A client never persists the seed and only holds it in memory to derive the keys as discussed in the Fudamentals.

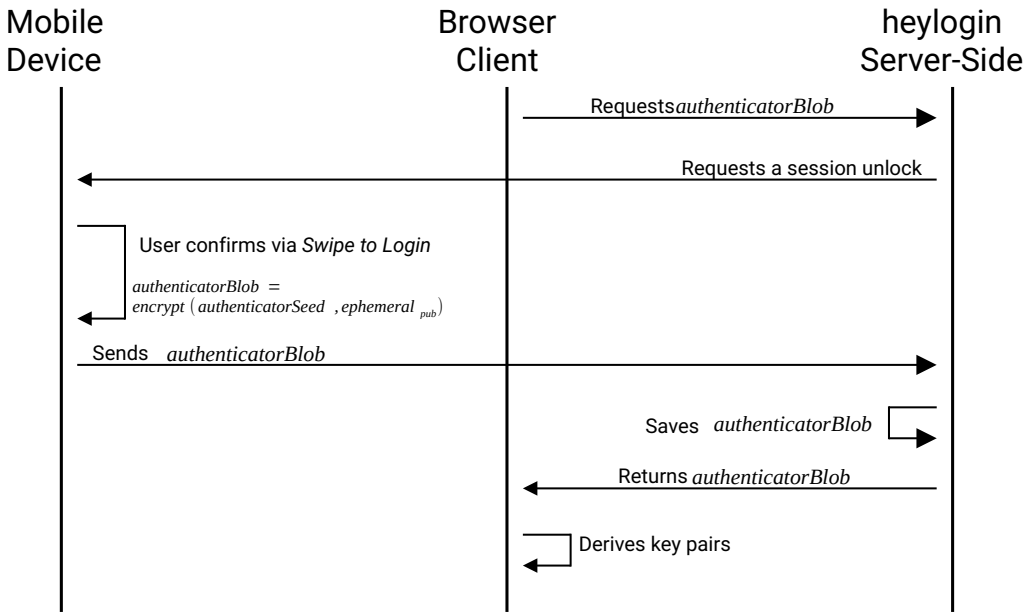

Figure 32: Unlocking a device

In practice, unlocking a session transfers the authenticator seed from the mobile device to the client. This is done by retrieving the sessions public key from the personal meta vault, encrypting the authenticator seed to that key and sending it to the server side which stores it. The session can then retrieve the encrypted authenticator seed, decrypt it with its session private key and derive all other keys. The encrypted authenticator seed on the server side can be deleted at will from inside the mobile app. It is also automatically deleted by the server side after 30 hours.

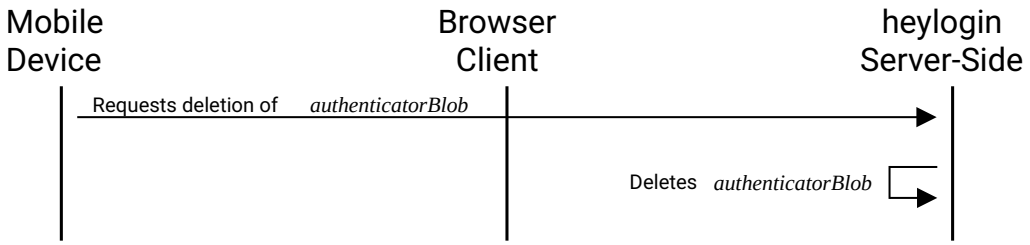

Figure 33: Locking a device

An unlocked session might persist any key pair that is *storable*. In practice, these are the $(profileSeedEnc^s_{sec}, profileSeedEnc^s_{pub}), (vaultKeyEnc^s_{sec}, vaultKeyEnc^s_{pub})$

and $(sig^s_{sec}, sig^s_{pub})$ key pairs. This allows a locked client to decrypt the encrypted $vaultKey^s$ which in turn allows it to parse logins but *not* their password, private custom fields or the TOTP secret. This property is used by the extension to show the overlay on websites with known logins. As soon as the user clicks on a specific login to log into the website, the extension will request an unlock so it can obtain the authenticator seed, derive all other high security key pairs, decrypt the high security profile seed, derive all other high security profile key pairs, delete the local decrypted copy of all seeds, decrypt the encrypted $vaultKey^{hs}$ and then decrypt the password to finish the login.

6.5 Organizations

The main use of profiles is in heylogin *Organizations*. Organizations are a way to manage access of multiple users to multiple vaults called *Teams*. Every member of an organization has exactly one *Organization Profile* which represents the membership of the user in that organization. The organization profile of a user is also the profile that is used to facilitate access to the organization's teams. Each team is a normal vault which simply has many Vault-Profile-Locks each storing the same vault encryption keys for a different profile.

6.5.1 Organization administrators

Organizations are managed by *Administrators*, or *Admins*, for short. Each organization has at least one admin. Being an admin allows a user to manage membership of other users inside the organization. The admin perfoms these actions by using the *Organization Admin Profile*.

Each organization possesses an admin profile which represents the administrator of the organization as only this profile is able to perform actions such as adding or removing members to and from the organization as only this profile is able to create Vault-Profile-Locks for all other profiles of the organization. The admin profile is ensured to have membership in all other vault of its organization.

From a cryptography view, the organization admin profile is the only administrator of an organization. Organization admins are therefore normal users who have a *Profile-Profile-Lock* for the admin profile which grants them access to the profile seeds of the admin profile. Cryptographically, the chain therefore goes from authenticator, to organization profile, to organization admin profile to any vault of that organization. Each organization must have at least one organization admin so that access to the organization admin profile is always possible.

6.5.2 Onboarding organization users

To add a new user to an organization, the admin first creates an organization profile for that user. The profile is initially not connected to the user via locks. Instead, the profile is in a *free* state. There can be, however, *empty* Vault-Profile-Locks for that profile to indicate membership in a vault as it is already possible to assign this profile membership. However, since the profile does not have any key material, the locks are empty and only serve as an membership indicator.

When creating such a *free profile*, a random *start code* is associated to the free profile. When the new member later wants to *claim* this profile, they will need to enter the start code as their authorization for claiming the profile. The start code is a six character random code consisting of a two character prefix and a four character suffix separated by a dash like so: `HL-A1B2`. The start code is visible to the organization admin and need to be communicated to the user in some other way.

It does not matter whether the user that should join the organization is already using heylogin or not. If they are not yet using heylogin, they will be able to claim the profile during account setup on their mobile device. If they are already using heylogin, they can claim additional profiles through the web app. A profile is *claimed* by the user by creating profile seeds and then creating the necessary Profile-Authenticator-Locks for that profile. This allows the user to access everything the profile has access to, which is currently nothing. The profile new has to be *connected*.

Connecting a profile involves the organization admin as they have to create the Profile-Authenticator-Locks for the user after the user has claimed the profile. This is easily done by the admin by creating the necessary Vault-Profile-Locks and replacing the empty locks with them. The admin does not need knowledge of the profile seed as they can access the public keys of the profile.

Should a user enter a wrong start code, the backend will record a failed claim attempt. After three failed claim attempts, the start code is blocked and all further attempts, even with the correct start code will result in failure to claim the profile. The organization admin has to generate a new random start code to reset the blocked state. This mechanism allows the start code to be short as only three guesses are possible until an admin action is needed.

6.5.3 Use FIDO2 devices without smartphone

For organizations, heylogin offers a way for users without smartphones to still use heylogin by using a FIDO2 authenticator instead. As described in the section about *FIDO2 devices as additional authenticators*, the PRF extension is not always available and heylogin offers a

workaround. This workaround is also needed to onboard security key only users as PRF cannot be used during the registration process. This is a limitation of PRF, not of heylogin. However, after registration, the security key can be "upgraded" to use PRF instead.

To onboard a standalone FIDO2 authenticator user, the organization admin first invites the user with their email address as normal. This creates a free profile as described previously. The user can now open the start page of the web app into which they enter their email address. This client then asks the server side for active invites. When an active invite is found, the organization name is shown to the user and the user is asked for their start code.

After entering and sending the start code, the server side checks whether the start code matches. If so, the server side answers with a challenge to be forwarded to the FIDO2 authenticator. The FIDO2 authenticator signs the challenge and returns it together with its public key and credential to the client. Furthermore, the web app generates an authenticator seed for us and encrypts it with the session key pair. The web client now behaves like a mobile device and derives the needed key pairs, generates preferences and inbox profile and their locks and sends them together with the email address, start code, the encrypted authenticator seed as well as the FIDO2 signed challenge, public key and credential to the server side.

The server side can now set up a user with a *Session Unlock Authenticator* as their only authenticator, a preferences and inbox profile including locks and a session with the encrypted authenticator seed for unlocking purposes saved. The encrypted authenticator seed will never be deleted on the server even if the session is locked as it cannot be recovered.

The client side does not persist the authenticator seed. It is only stored in encrypted form on the server side and only accessible by the session that created it. Losing access to the session, e.g. by deleting the browser local storage, completely shuts out the user and recovery via the organization admin is required. This also limits the account to one session only as the encrypted seed cannot be transferred to a new session. Recovery of the account is only possible by an organization administrator.

If the PRF extension is available, these FIDO2 devices can be upgraded as explained in the section about *FIDO2 devices as additional authenticators*.

6.5.4 Account recovery

Account recovery is needed when a user has lost access to their push authenticator, e.g., because their mobile device broke down. In such a case, there are several ways an account can be restored. The most recommended way is by informing an organization admin. Organization admins can then disconnect the old push authenticator of the user and connect a new one. Disconnecting the old push authenticator does not delete the profile associated to that user, so no

data is lost when disconnecting / connecting a push authenticator. It only deletes the Profile-Authenticator-Locks of the user for that profile and clears the {Vault,Profile}-Profile-Locks. When a new push authenticator should be connected to the profile, the user first regenerates the profile to a *claimed* state by generating new profile seeds and creating the Profile-Authenticator-Locks. An organization admin can then replace the empty {Vault,Profile}-Profile-Locks with the correct ones. While disconnected, the organization profile is considered a free profile.

Alternative ways to restore an account is by using the backup code authenticator or the platform backup authenticator. The platform backup authenticator works by accessing the seed stored inside a backup. By switching to a new mobile device, this authenticator is automatically made available as soon as the cloud backup has been restored. When either the platform backup or backup code authenticator are used, the server side will remove the push authenticator and all its locks. The new mobile device has to generate a new push authenticator seed and derive the necessary keys to register it as a new push authenticator with a new Profile-Authenticator-Lock. When using a platform backup or backup code authenticator, the server side only allows for replacing the primary authenticator with a new one. Other operations are denied.

Afterwards, the new push authenticator will regenerate all profiles by generating new profile seeds and replacing all Profile-Authenticator-Locks and all Vault-Profile-Locks. Then, all vaults are flagged as *dirty* to be regenerated before the next commit is made.

6.6 Key rotation

heylogin's usage of profiles enables key rotation on demand, called *regeneration* in previous sections. Typically, vaults and profiles are regenerated, i.e., their keys are replaced with new keys and their content is re-encrypted, when it is cryptographically necessary to do so. This is mainly the case when we want to ensure post-compromise security, i.e., an attacker that has access to old key material who then gains access to new encrypted data must not be able to decrypt this data with the old key material.

6.6.1 Vaults

Vaults are regenerated when they are flagged as *dirty* and a client tries to create a new commit. Vaults get flagged as dirty when a user who used to have access should not have access any more.

A common case is a team inside an organization. Users Alice and Bob have access to the Team vault Accounting with five logins. Alice removes Bob from the team which flags the vault as dirty.

As long as Alice is not changing anything inside the vault, e.g. creating new logins, deleting others, or changing data of existing logins, the vault's keys are not rotated. Only when Alice changes something, the keys are also changed. If Bob were to gain access to the encrypted vault data before the key rotation, he would be able to decrypt it with his old key material, however, this is fine as he will gain no new information he would not have had already from when he was still a member of the team.

Vault keys are rotated as follows:

The mobile device will regenerate the vault keys of a vault by first squashing all current commits into a new, single commit that holds the most up-to-date state. This new commit marks the start of a new *Generation* in the vault. All previous commits are discarded from the server. The new commit is encrypted with a new $vaultKey^{s}$ and its sensitive contents with a new $vaultKey^{hs}$ which are encrypted and stored inside new Vault-Profile-Locks for that vault. Old locks are discarded.

6.6.2 Profiles

Profiles are regenerated when necessary as outlined in *Account recovery*.

Regeneration entails generating new profile seeds and then replacing all Profile-Authenticator-Locks as well as all Vault-Profile-Locks and, if present, all Profile-Profile-Locks. In organization scenarios, when claiming a profile, regeneration is a two step process involving the user and an organization admin. The user only generats the seeds and Profile-Authenticator-Locks while the organization admin can then create the Vault-Profile-Locks as outlined in *Account recovery*.

6.6.3 Authenticators

Authenticators are typically not regenerated but rather replaced with a new authenticator of the same type. A user can easily replace their Webauthn authenticator by removing and then re-adding their FIDO2 device and their backup code authenticator by creating a new backup code. Replacing other authenticators only happens in the account recovery scenario outlined in *Account recovery*.

6.6.4 Cryptographic Agility

heylogin's modular architecture of authenticators, profiles and vaults not only makes it possible to easily rollout key changes to rotate them but also to change the used algorithms if needed. Should any of heylogin's used cryptographic algorithms need to be changed, e.g., to change to

a post-quantum secure version, clients could successively update profiles and vaults on their own.

Appendices

Appendix A: Threat model limitations

This appendix documents limitations, assumptions, and threat scenarios that are intentionally excluded from the scope of heylogin's threat model. Its purpose is to clearly define the trust boundaries of the system, highlight conditions under which the security guarantees no longer apply, and prevent misinterpretation of the threat model's scope or intent.

Compromised browser or device

Threats resulting from a fully or partially compromised end-user device or browser environment are out of scope for this threat model. This includes malware operating within the browser or operating system context, such as infostealers, that execute with the same or higher privilege level as the heylogin web application.

As described in *Protection against hardware keyloggers*, heylogin eliminates entire classes of credential theft by avoiding password entry and by limiting the lifetime and reuse of sensitive secrets. Likewise, as outlined in *Security of passwords in memory*, the exposure of sensitive data in memory is minimized to reduce the attack surface for memory scraping techniques.

However, malware such as infostealers can directly access browser-managed artifacts including session cookies, local storage entries, or in-memory authentication state. By extracting these artifacts, an attacker may abuse an existing authenticated session without needing to defeat cryptographic protections, authentication protocols, or end-to-end encryption mechanisms.

Such attacks assume a loss of integrity of the client environment and bypass application-level safeguards by operating outside the application's trust boundary. While heylogin's design reduces the impact of certain malware techniques, it cannot fully protect against adversaries that already control the user's browser or device.

Accordingly, attacks involving session hijacking, cookie theft, browser injection, keylogging, memory scraping, or unauthorized access to local storage on a compromised device are explicitly excluded from the scope of this threat model.

Cryptographic operations in the browser

The heylogin web application is delivered to the client exclusively over HTTPS. The end-to-end encryption scheme described in this whitepaper is fully implemented within the web application hosted at heylogin.app.

While the system architecture may appear to allow server-side access to encrypted data, heylogin servers are not capable of decrypting user data. All cryptographic operations related to decryption are performed locally within the client environment after the application code has been fetched. Encrypted data is processed and stored within the client's local storage, and plaintext is never exposed to server-side components.

The authenticity and integrity of the web application code depend on the security of the TLS connection over which it is delivered. Attacks that compromise TLS, certificate authorities, or the browser trust model could lead to the delivery of a modified client and are therefore considered out of scope for this threat model.

Attacks on app and extension stores

The authenticity and integrity of the heylogin app and browser extension code depend on the security and honesty of the respective app and extension stores as well as the underlying operating system or browser.

While platform-level protections exist, such as preventing app upgrades signed with a different signing key on Android, these mechanisms rely entirely on store and platform security and primarily protect against unauthorized third-party distribution. Attacks assuming a compromised official store, signing infrastructure, or delivery channel are considered out of scope for this threat model.

Attacks on platform backups

Similar to the considerations outlined in the previous section, heylogin supports platform provided backup authenticators for account recovery, as described in *Platform Backup Authenticator*. This mechanism relies on the operating system's native key storage and cloud backup infrastructure.

On both iOS and Android, platform backups are configured to be only transmitted and stored end-to-end encrypted, provided that Apple and Google correctly implement their documented security architectures and act honestly. On iOS, the backup authenticator is stored in iCloud Keychain. On Android, it is stored in Android Keystore and included in the device's configured

backup system, typically Google Cloud Backup. The security properties of these backups are enforced by the platform vendor and are outside the direct control of heylogin. Attacks targeting platform backups or the associated cloud accounts are therefore considered out of scope for heylogin's core end to end encryption model. The use of the Platform Backup Authenticator is optional. Users and organizations that do not trust platform cloud backups should disable them at the operating system level.

Attacks on security chips

Attacks that assume the successful compromise of hardware-backed security components, such as secure elements, TPMs, security keys, or trusted execution environments, are considered out of scope for this threat model.

Advanced attacks against elliptic curve cryptography (ECC) operations in security chips represent a highly specialized threat class. A notable example is the EUCLEAK side-channel attack, which demonstrated that electromagnetic leakage from Infineon-based YubiKey 5 Series devices could be used to extract private keys given physical access and specialized measurement equipment[35][36]. More recently, research has shown that, under specific conditions, security chips can be attacked by escalating control from a compromised operating system into a trusted execution environment on affected devices, as demonstrated in work presented at 39C3[37]. In addition, a protocol issue (CVE-2025-29991) was identified in YubiKey firmware, but due to existing mitigations and the complexity of exploitation, it is considered low risk[38].

Despite these findings, hardware-backed security remains significantly more resilient than master-password-based systems. Master passwords can be attacked remotely and at scale through phishing, credential stuffing, malware, or server-side breaches. In contrast, attacks on security chips typically require physical access to the device, deep technical expertise, and often lab-grade equipment, making large-scale remote exploitation impractical.

To reduce residual risk, organizational measures are recommended, including the use of secure and supported devices that receive regular security updates and the mandatory use of trusted security keys with up-to-date firmware (for example, YubiKey 5 Series v5.7.0+ or devices using cryptographic libraries updated after EUCLEAK). These measures help limit the practical impact of hardware-level attacks beyond the heylogin security design.

[35] Yubico. Security Advisory YSA-2024-03 - Infineon ECDSA Private Key Recovery
[36] NinjaLab. EUCLEAK: Extracting Secrets from European eID Smartcards, https://ninjalab.io/eucleak/
[37] Philipp Mao, Marcel Busch, Mathias Payer. Not To Be Trusted - A Fiasco in Android TEEs, https://nebelwelt.net/blog/2025/1227-fiasco.html
[38] Yubico. Security Advisory YSA-2025-02 - FIDO PIN/UV Auth Protocol Two Out of Conformance

Public key verification

heylogin does not implement user-facing public key verification. There is no mechanism to display, compare, or manually verify key fingerprints, such as those used in SSH, OpenPGP, or Signal. To the best of our knowledge, this limitation is shared by existing password managers and similar systems.

As a result, heylogin servers could theoretically deliver an incorrect public key during key distribution, for example when encrypting shared vaults for multiple users, thereby acting as a man-in-the-middle. Attacks that rely on malicious or incorrect public key distribution are therefore considered out of scope for this threat model.

Post-quantum security

heylogin follows a post-quantum strategy, but is currently not fully post-quantum secure. This is because the asymmetric cryptographic mechanisms in use are based on Curve25519, which is not resistant to potential future attacks using a quantum computer.

At the same time, heylogin already addresses relevant post-quantum threat scenarios. In particular, mitigations against *harvest now, decrypt later* attacks are considered, where encrypted data is collected today with the intention of decrypting it in the future once sufficiently powerful quantum computers become available[39]. One key mitigation is the gradual adoption of post-quantum-secure or hybrid transport encryption mechanisms.

heylogin's key rotation mechanism allows post-quantum cryptography to be introduced into the end-to-end encryption architecture at a later stage. We continuously monitor the evolution of post-quantum cryptographic algorithms, their standardization, and the practical threat landscape.

BSI TR-02102-01 compliance

heylogin uses Curve25519, XSalsa20, Poly1305, and Argon2 for its cryptographic architecture. Argon2 is recommended in the BSI Technical Guideline TR-02102-1 (version 2025-01)[40], while Curve25519, XSalsa20, and Poly1305 are not specifically listed. The guideline does not claim completeness, and omission from the document does not imply that an algorithm is insecure.

[39] Wikipedia, Harvest now, decrypt later, https://en.wikipedia.org/wiki/Harvest_now,_decrypt_later

[40] BSI TR-02102-1 "Cryptographic Mechanisms: Recommendations and Key Lengths" Version: 2025-1, https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Publications/TechGuidelines/TG02102/BSI-TR-02102-1.html

The BSI acknowledges in a public response provided under the German Freedom of Information Act that no cryptographic weaknesses in X25519 are known[41].

The algorithms were selected as modern, well-researched cryptographic primitives with a strong security track record. Alignment with BSI TR-02102-1 is treated as cryptographic guidance rather than a strict compliance requirement within this threat model.

[41] FragDenStaat, Anfrage zu Elliptische Curve X25519, https://fragdenstaat.de/anfrage/elliptische-curve-x25519/