Dashlane's Security Principles & Architecture

DASHLANE

May 16, 2023

v2.0.3

Contents

Figures

1 General Security Principles

1.1 List of secrets

Dashlane uses many secrets to secure user's data. Some of them are described as follows:

Key Name	Key Symbol	Description
User Master Password	$User_{MP}$	Password/Passphrase generated by the user to derive the key to encrypt the user's vault. The User Master Password is expected to be as random as possible
Intermediate Key	$Intermediate_{Key}$	Random 32-byte key, generated by local devices
User Device Key	$Device_{Key}$	Random 32-byte key, generated by local devices
User Secondary Key	$UserSecondary_{Key}$	Random 32-byte key, generated server side for 2FA usage

Table 1: Dashlane Security ArchitectureDashlane Secrets Overview

1.2 Protection of User Data in Dashlane

Protection of user data in Dashlane relies on 3 separate secrets:

- The **User Master Password**:

    ▷ It is never stored on Dashlane servers, nor are any of its derivatives (including hashes).

    ▷ By default, it is not stored locally on the disk on any of the user's devices; we simply use it to decrypt the local files containing the user data.

    ▷ It is stored locally upon user request when enabling the feature "Remember my Master Password."

    ▷ In addition, we ensure that the user's Master Password is never transmitted over the internet. [1]

- The **Intermediate Key**: in some cases (local storage), we use $Intermediate_{Key}$ encrypted with a derivative of $User_{MP}$.

- The **User Device Key**s: unique key for each device enabled by a user:

    ▷ Auto-generated for each device.

    ▷ Used for authentication.

[1] The only derivative of it that is sent over the internet is the final encrypted vault. The following paragraphs outline how we ensure its resilience to attacks.

1.3 Local Access to User Data

Access to the user's data requires using the $User_{MP}$, which is only known by the user. It is used to generate the symmetric Advanced Encryption Standard (AES) 256-bit key for encryption and decryption of the user's personal data on the user's device.

We use Web Crypto API for most browser-based cryptography and the native libraries for iOS and Android. We use the Argon2 reference library compiled into

Web Assembly (Wasm) or linked to the mobile app.

1.4   Local Data Usage After Decrypting

Once the user has input their $User_{MP}$ locally in Dashlane and their user data has been decrypted, data is loaded in memory.

The Dashlane client operates within significant constraints to use decrypted user data effectively and securely:

- Dashlane processes access individual passwords to autofill them on websites or to save credentials without having to ask the user for $User_{MP}$ each time.

- The Argon2d (or PBKDF2) derivation used to compute the AES keys adds significant latency (the purpose of this is to protect against brute force attacks).

  See paragraph Memory Protection for more on memory management.

1.5   Use of 2FA Applications to Increase User Data Safety

At any time, a user can link their Dashlane account to a 2FA application on their mobile device (we recommend using Dashlane Authenticator, or alternatives such as Google Authenticator). All of their data (both the data stored locally and the data sent to Dashlane servers for synchronization purposes) is then encrypted with a new key, which is generated by a combination of $User_{MP}$ and a randomly generated key $UserSecondary_{Key}$ stored on the Dashlane server, as described in the following steps:

- The user links their Dashlane account with their 2FA application.

- Dashlane servers generate and store $UserSecondary_{Key}$, which is sent to the user's client application.

- All personal data are encrypted with a new symmetric AES-256 bit key generated client-side from both $User_{MP}$ and $UserSecondary_{Key}$.

- $UserSecondary_{Key}$ is never stored locally.

- The next time the user tries to log into Dashlane, they will be asked by Dashlane servers to provide a One-Time Password generated by the 2FA application. Upon receiving and verifying this One-Time Password, Dashlane servers will send the $UserSecondary_{Key}$ to the client application, allowing the user to decrypt their data.

User data can be decrypted only by having both $User_{MP}$ and the 2FA application linked to the user's account.

1.6   Authentication

As some of Dashlane's services are cloud-based (data synchronization between multiple devices, for instance), there is a need to authenticate the user on Dashlane servers.

Authentication of the user on Dashlane servers is based on $Device_{Key}$ and has **no relationship with the User Master Password**.

When a user creates an account or adds a new device to synchronize their data, a new User Device Key is generated by the servers. $Device_{Key}$ is composed of 40 random bytes generated using the OpenSSL RAND_byte function. The 8 first bytes are the access key, and 32 remaining bytes are the secret key.

$Device_{Key}$ is received by the user's device and is stored locally in the user data, encrypted as all other user data, as explained earlier. On the server side, the secret key part is encrypted so that employees cannot impersonate a given user device. When a user has gained access to their data using $User_{MP}$, Dashlane is able to access $Device_{Key}$ to authenticate them on our servers without any user interaction.

As a result, Dashlane does not have to store $User_{MP}$ to perform authentication.

1.7   Communication

All communications between the Dashlane application and the Dashlane servers are secured with HTTPS. HTTPS connections on the client side are performed using OpenSSL. [2]

[2] Cloudflare certificate - ASN1 OID: prime256v1; Signature Algorithm: ecdsa-with-SHA256

The HTTPS communication between the Dashlane application and the Dashlane servers use SSL/TLS connections.

Main TLS protocol steps are as follows:

- The client and the server negotiate to choose the best cipher and hash algorithm available on both sides.

- The server sends a digital certificate.

- The client verifies the certificate by contacting a Certificate Authority.

- The client encrypts a random number with the server's public key and sends it to the server.

- The server decrypts this number, and both sides use this number to generate a symmetric key, which is used to encrypt and decrypt data.

1.8   Details on Authentication Flow

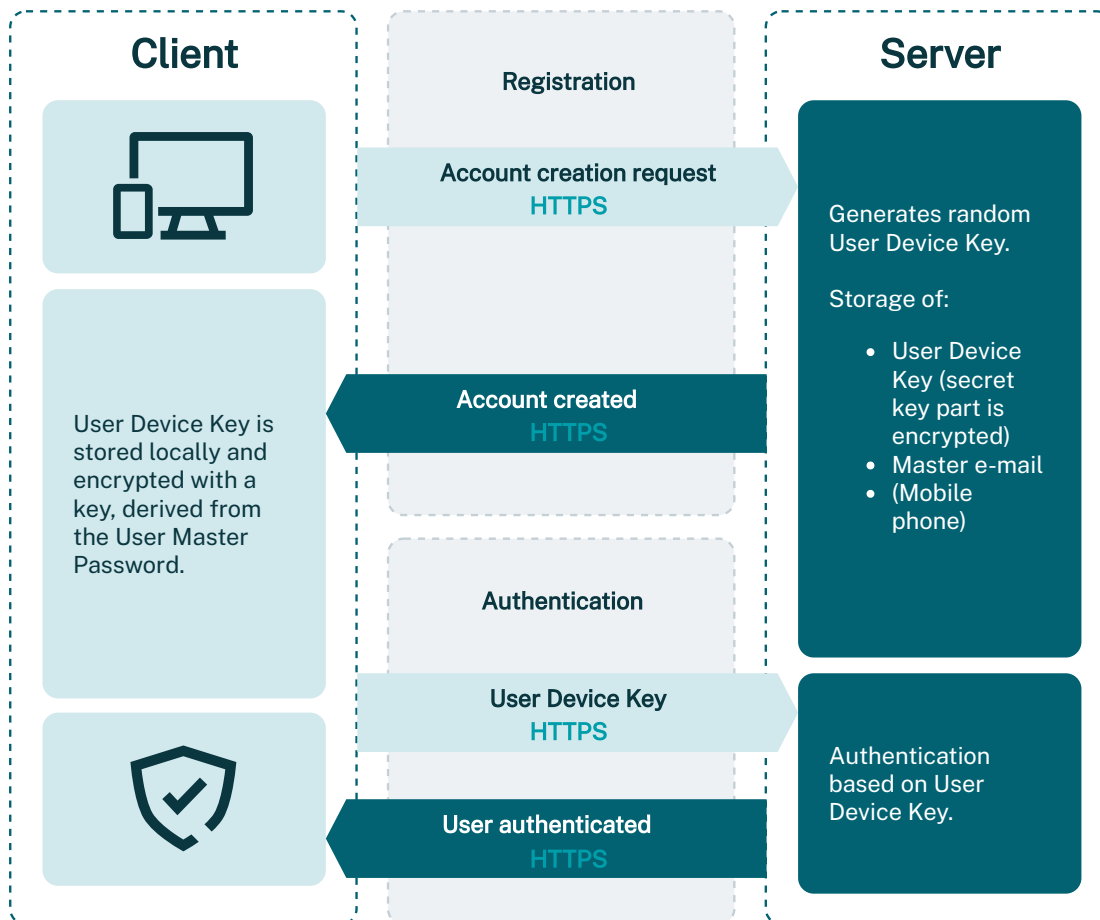The initial registration for a user follows the flow described in Figure 1.

Figure 1: Authentication Flow During Registration

As seen in Figure 1, $User_{MP}$ is never used to perform server authentication, and the only keys stored on our servers are the User Device Keys.
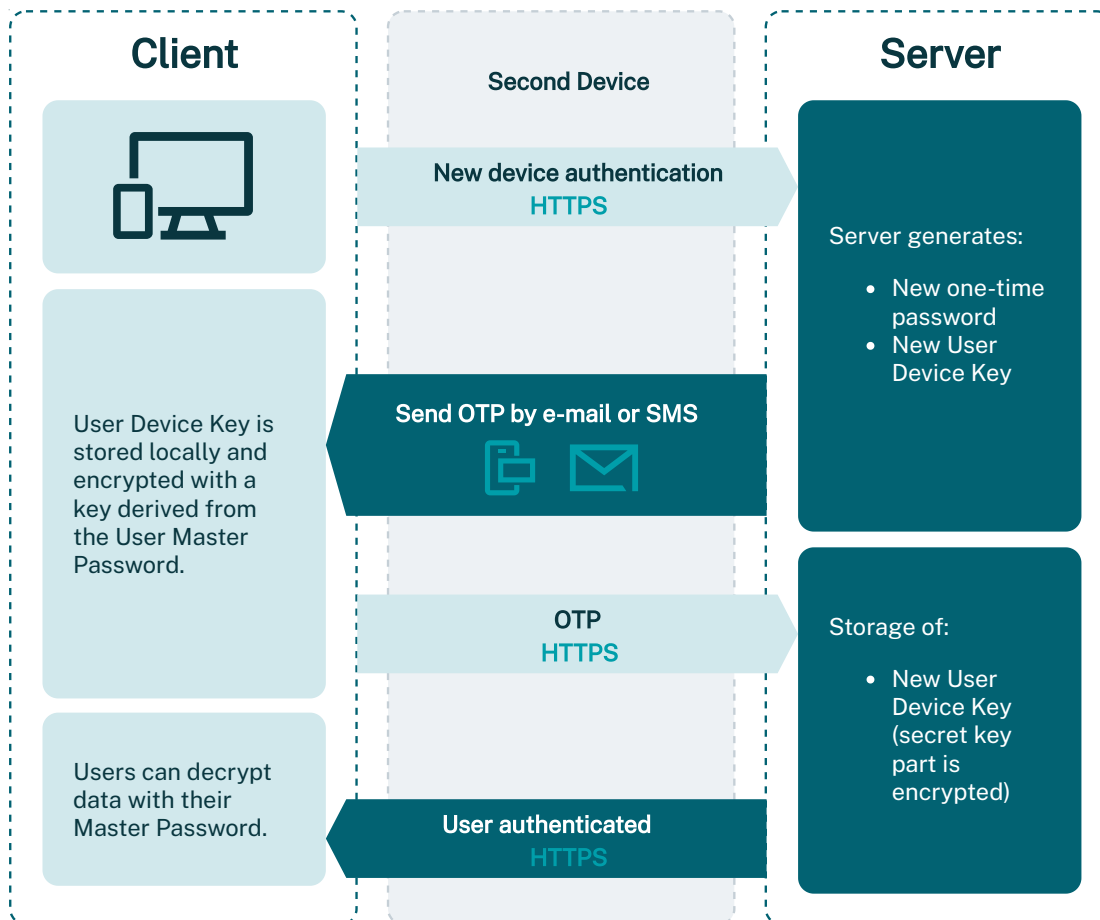
Figure 2: Authentication When Adding a New Device

When a user adds an additional device, Dashlane needs to make sure that the user adding said device is indeed the legitimate owner of the account. This is to gain additional protection in the event $User_{MP}$ has been compromised and an attacker who does not have access to their already-enabled device is trying to access the account from another device.

As shown in Figure 2, when a user is attempting to connect to a Dashlane account on a device that has not yet been authorized for that account, Dashlane generates a One-Time Password (a token) that is sent to the user either to the email address used to create the Dashlane account initially or by text message to the user's mobile phone if the user has chosen to provide their mobile phone number.

To enable the new device, the user has to enter both $User_{MP}$ and the token. Only once this two-factor authentication has been performed will Dashlane servers start synchronizing the user data on the new device. All communication is handled with HTTPS, and the user data only travels in AES-256 encrypted form. Please note again that $User_{MP}$ never transmits over the internet.

1.9   Keeping the User Experience Simple
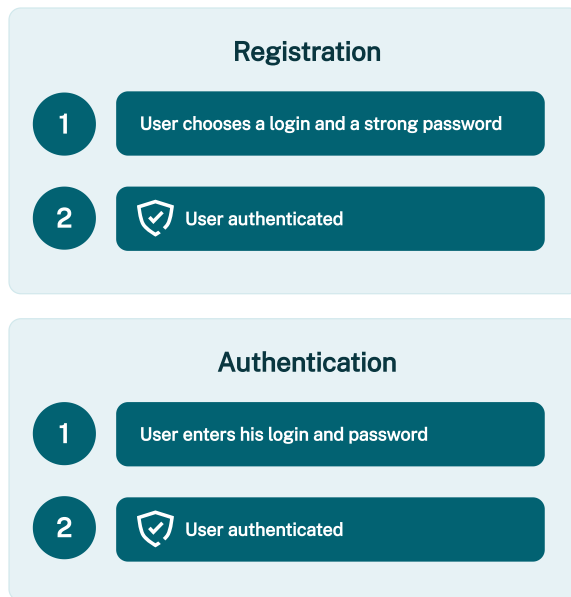
Figure 3: Registration and Authentication Steps

All along, our goal has been to keep the user experience simple and to hide all the complexity from the user. Security is growing more and more important for users of cloud services, but they are not necessarily ready to sacrifice convenience for more security.
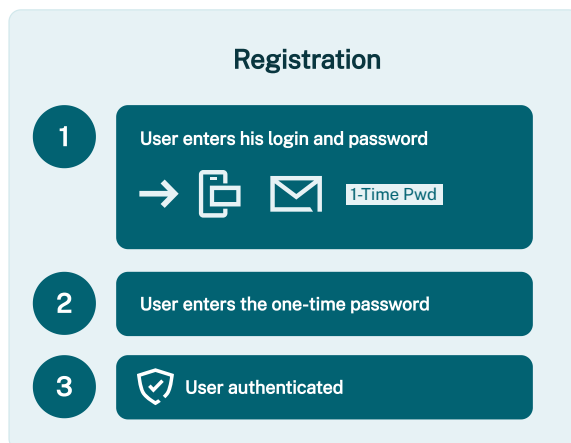
Figure 4: Second Device Registration Steps

Even though what goes on in the background during the initial registration steps is complex (see Figure 3), the user experience is very simple. All they have to do is pick a (strong) $User_{MP}$, and all the other keys are generated by the application without user intervention.

When adding an additional device, the process is equally simple while remaining highly secure through the use of two-factor authentication described in Figure 4.

1.10 Use of 2FA Application to Secure the Connection to a New Device

At any time, a user can link their Dashlane account to a 2FA application on their mobile device. When they attempt to connect to a new device, instead of sending them a one-time password by email, Dashlane asks the user to provide a one-time password generated by the 2FA application.

After receiving and verifying the one-time password provided by the user, Dashlane servers will store the $Device_{Key}$ generated by the client application, as described in Figure 4.

1.11 2-Factor Authentication

Dashlane offers 2-factor authentication that can be activated from the security settings in the web extension or mobile app to force the usage of a second factor each time the user logs into Dashlane.

Supported two-factor methods include 2FA applications such as Google Authenticator or U2F-compatible devices such as Yubikeys. U2F is an open protocol from the FIDO Alliance (https://fidoalliance.org). Dashlane is a board-level member of the FIDO Alliance.

1.12 Sharing Data Between Users

Dashlane allows users to share credentials and Secure Notes with other users, or with groups of users, in such a way that Dashlane never directly accesses a user's data at any point. In fact, Dashlane's servers never have access to the content of shared data.

Dashlane's sharing relies on asymmetric encryption; upon account creation, a unique pair of public and private RSA keys are created by the Dashlane application for each user. The private key is stored in the user's personal data, and the public key is sent to Dashlane's servers. RSA public and private keys are generated using the OpenSSL function RSA_generate_key_ex, using a key length of 2048 bits, with 3 as a public exponent.

Here is the process for a user, Alice, to share a credential with another user, Bob:

- Alice asks Dashlane's servers for Bob's public key.

- Alice generates a 256-bit AES key using a cryptographically secure random function. This key is unique for each shared item and is called an ObjectKey.

- Alice encrypts the ObjectKey using Bob's public key, creating a BobEncryptedObjectKey.

- Alice sends the BobEncryptedObjectKey to Dashlane's servers.

- Alice encrypts her credential with the ObjectKey, using AES-CBC and HMAC-

SHA2 to create an EncryptedCredential.

- Alice sends the EncryptedCredential to Dashlane's servers.

- When Bob logs in, Dashlane's servers inform him that Alice wants to share a credential with him. Bob must manually accept the item in his Dashlane application and sign his acceptance using his private key.

- Upon acceptance, Dashlane's servers send Bob the BobEncryptedObjectKey and the EncryptedCredential.

- Bob decrypts the BobEncryptedObjectKey with his private key and gets the ObjectKey.

- Bob decrypts the EncryptedCredential with the ObjectKey and adds Alice's plain text credential to his own personal data.

Sharing data with a group of users follows the same security principle: Use a user's RSA public and private keys to send protected AES keys, sign a user's action, and use intermediary AES keys to exchange data.

To summarize:

- Each user has a pair of public and private RSA 2048-bit keys:

  ▷ Public keys are used to encrypt information only a specific user can decrypt.

  ▷ Private keys are used to sign actions users are performing.

- For each credential or secure note shared, an intermediary AES 256-bit key is created and used to perform data encryption and decryption.

1.13   Account Recovery

Account recovery allows Dashlane Business users to regain access to Dashlane by resetting $User_{MP}$. Our patented process preserves zero-knowledge. Through account recovery, master passwords are never stored on any servers nor transmitted in any form.

Our solution allows users to reset $User_{MP}$ and recover the data stored on an authorized device. Account recovery is an optional feature admins can activate for their Dashlane Business account in the Admin Console.

To enable recovery, the user's local key —itself encrypted with $User_{MP}$ —is also encrypted using a unique user recovery key, which is generated and used for all of the user's devices when they opt into account recovery. This user recovery key is then encrypted using a unique server-side recovery key, which is only known to Dashlane and the user's client devices. When an admin enables account recovery, their public key is used to encrypt the server-side recovery key, which as aforementioned, was already used to encrypt the user's recovery key. An admin can then, via

their private key, later access the user's recovery key protected by the server-side recovery key.

When a user requests account recovery, they are asked to verify their account and create a new $User_{MP}$. A critical step of the recovery process is the verification of the identity of the user. It is up to the admin, acting as a trusted third party, to ensure the user requesting recovery is indeed the owner of the account. If an admin approves the request, the server-side recovery key, which protects the user's recovery key, is securely exchanged from the admin to the user through a public/private key system. On the user's device, the user's recovery key is then decrypted using the server-side recovery key, provided by Dashlane after the user's identity and request have been validated. The user's recovery key is then used to decrypt the user's local key, which in turn is used to decrypt the user's data. The recovered data is then re-encrypted with $User_{MP}$ and re-synced to the Dashlane servers.

As this process involves a master password change, all of the user's devices have to be registered once again to Dashlane for the user to access their newly encrypted data.

Important privacy note: the account recovery process relies on the admin being a trusted third party. In case the Dashlane admin has access to both the user's device and the user's email used as a Dashlane account, the admin would be in a position to trigger an account recovery from the user's device and get access to the user's vault and personal data.

1.14 Dark Web Monitoring for Master Password

This feature allows Dashlane users to be alerted if their master password or an employee's master password has been identified in a data breach. To check if the master password of a user is compromised, we are going to check if it is present in the databases resulting from the various data leaks that we collect. The data provider we work with is Spycloud. We collect the data through API requests, and transform all data into hashes using the Argon2 function before storing them on our servers. When a user enters his master password on his mobile or Web application, we start by transforming it using the Argon2 function and a salt [3] present in the client application, giving us a 32 bytes long hash.

[3] The salt we use is specific for this feature and different from the one used to build the user's encryption key

Algorithm	Iterations	Mem. usage	Parallelism	Threads	Hash length
Argon 2d v1.3	3	32768	2	2	32

Table 2: Argon2 configuration

To respect our zero-knowledge architecture, we use a process called "K-anonymity" to guarantee that no one, not even Dashlane can access the master password. For this, the complete hash never leaves the user's device, but we only send the first three bytes of it to our servers and compare those bytes to the entries we have in our database. If we have one or more matches, we send the list to the users and finally, the application is able to make a complete comparison between the local hash and the one(s) coming from Dashlane's servers, and at the end, warn the user if his master password has been found in a data leak.
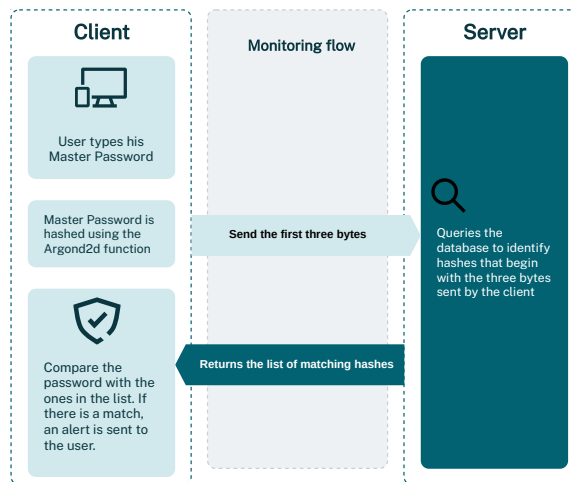
Figure 5: Dark Web monitoring for Master Password flow

2 Single Sign-On (SSO)

2.1 Introduction

Dashlane Business supports login with single sign-on (SSO), using any SAML 2.0 enabled IdP.

In a single-sign-on setup, the user doesn't have to input $User_{MP}$. Instead, a random key is generated at account creation. This key (the data encryption key) is delivered to the Dashlane app after the user successfully logs in to the IdP, and it is used as a symmetric encryption key to encrypt and decrypt the user data.

This section details how the key is stored and delivered to the user in order to make sure that the zero-knowledge principle is maintained.

2.2 General Principle

The integration of SSO with the Dashlane app requires an entity storing users' encryption keys and delivering them upon authentication. This entity has the knowledge of every user's key, so it's highly sensitive. Moreover, Dashlane can't host such an entity without more concerns because this would break our zero-knowledge principle by providing us access to the encryption keys of our users.

The previous entity in charge of users' encryption keys is called the Encryption Service and it could be hosted two different ways to follow our zero-knowledge rule:

- **Self-hosted:** the Encryption Service is a server deployed inside the infrastructure of Dashlane Business customer

- **Hosted in a secure enclave by Dashlane:** the Encryption Service is a service

running in Dashlane infrastructure, in a secure enclave to respect our zero-knowledge principle

2.3   Single Sign-On with the Self-Hosted Connector

2.3.1   Overview

To avoid storing all the keys in one place, the data encryption key is composed of 2 parts:

- 64 random bytes held by the Encryption Service

- 64 random bytes held by Dashlane's servers in the cloud

The Encryption Service is a server component that the customer operates (either in the cloud or on premises). It acts as the service provider in the SAML 2.0 flow. After a successful authentication to the Encryption Service using SAML, the first part of the key is delivered to the Dashlane client application along with a token that allows it to get the second part from the Dashlane server.

Once both parts of the keys are retrieved by the client app, they are XORed together, and the resulting 64 bytes are used as a symmetric key to encrypt and decrypt user data.

This system ensures zero-knowledge as the first part of the key and is only known by the Encryption Service and the client app, both of which are managed by the customer.

It also makes sure that a compromised Encryption Service cannot be used to fetch the keys of users without leaving traces on Dashlane servers (an API call to the Dashlane server is required to fetch the second part of the key).

2.3.2   Services

Dashlane Server/API (API) The servers operated by Dashlane in the cloud, where user data is stored encrypted.

Encryption Service (SP) A service acting as the service provider in the SAML 2.0 flow. The service is distributed by Dashlane, but it's hosted and managed by the customer on premises or in the cloud.

Identity Provider (IdP) The SAML 2.0 identity provider (e.g. ADFS, Azure AD, Okta) of the customer. This service is not provided by Dashlane. It is operated by the customer or by a third party.

2.3.3   Keys, secrets, and certificates

IdP key and certificate (IdP_{Key} **/** IdP_{Cert}**)** Public and private keys of the IdP. The private key is held by the IdP, while the certificate needs to be provided to the SP in the configuration file. It is used by the IdP to sign and by the SP to verify the SAML assertions.

Master SP Key / Encryption Service Key ($Master\_SP_{Key}$**)** A 64 bytes secret key, generated randomly by the Team Admin Console (client side). It is stored in the configuration file of the SP, and is only known by the Team Admin. It is used by the SP to encrypt/decrypt the $User\_SP_{key}$ before storing them in the API.

User SP Key ($User\_SP_{Key}$**)** A 64 bytes secret key, generated randomly by the SP. It is stored and encrypted in the API.

User Server Key ($Server_{Key}$**)** A 64 bytes secret key, generated randomly by the client. It is stored unencrypted in the API.

User vault key ($Vault_{Key}$**)** $User\_SP_{Key} \oplus Server_{Key}$. It is used by the client to encrypt/decrypt users' data before storing them in the API.
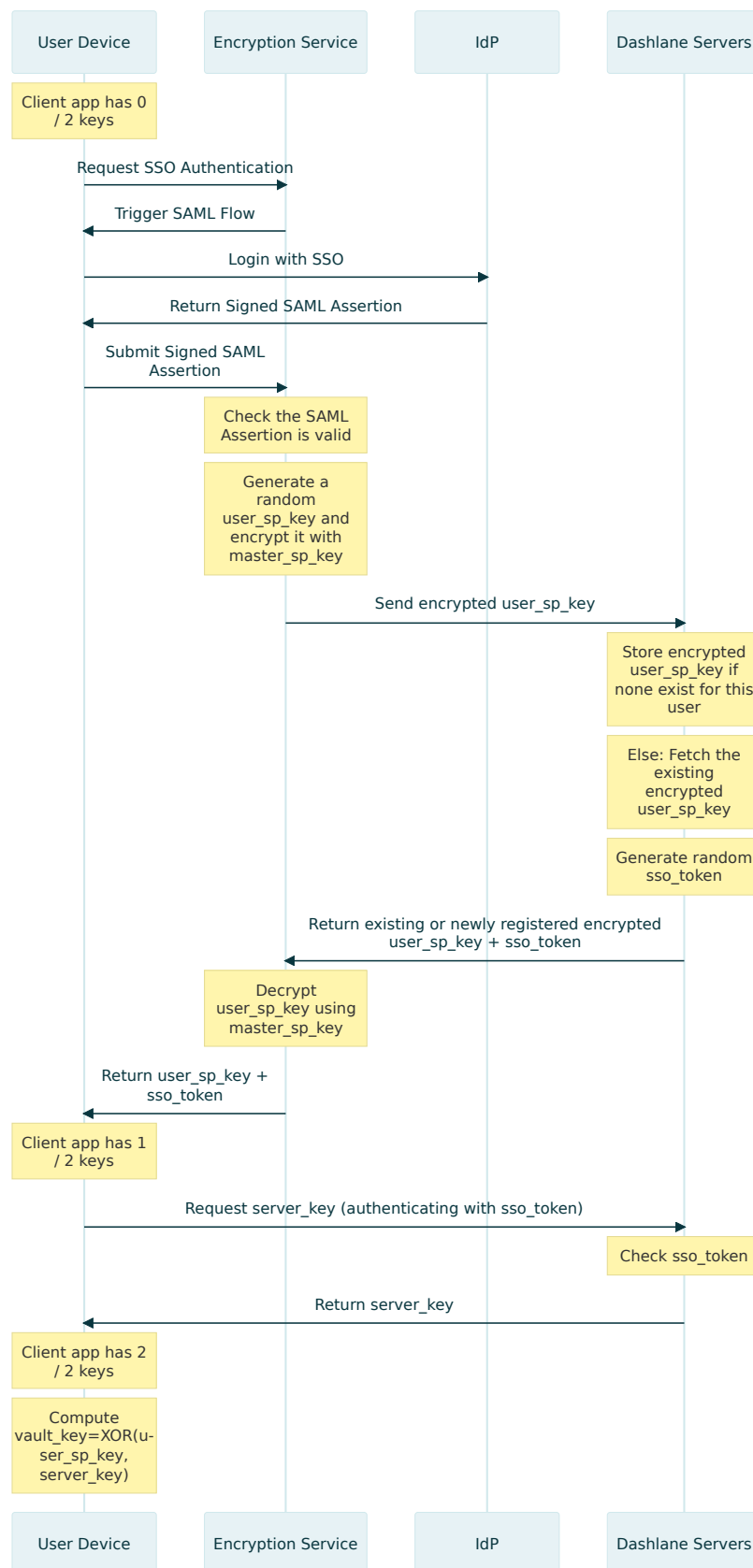
2.3.4   Workflow

Figure 6: Self-Hosted SSO Workflow

2.4   Single Sign-On with the Dashlane-Hosted Connector

2.4.1   Overview

In the Dashlane-hosted connector setup, the Encryption Service service is hosted and managed by Dashlane. To prevent Dashlane from accessing users' encryption key, breaking the zero-knowledge principle, the Encryption Service runs in a so-called secure enclave.

A secure enclave is a term coming from the field of trusting computing. This is the name given to an isolated computing unit or a Trusted Execution Environment (TEE). This technology provides a way to process data inside an environment that is not readable by any other process of the hosting machine besides the process running inside the enclave. Moreover, secure enclaves can generate attestation with the fingerprint of the code they run. This way, clients communicating with an enclave can get assurances of the code they are communicating with and decide if they trust this code to process their data.

Secure enclaves are just computing units with CPU and volatile memory resources. They are not provided with persistent storage. To circumvent this problem, a Key Management Service (KMS), which can authenticate that requests are coming from trusted enclaves, is required to encrypt the storage of secure enclaves.

Dashlane leverages secure enclave technology to run a Encryption Service service without being able to access users' encryption keys processed by the Encryption Service.

2.4.2   Cryptographic materials

Dashlane confidential SSO workflows require a lot of cryptographic keys and certificates defined in the table 3.

Key Name	Key Symbol	Description
Enclave Master Key	EM_{Key}	Key generated and stored within the KMS in order to encrypt/decrypt EL_{Key}
Enclave Local Key	EL_{Key}	Key generated within the KMS at the first enclave bootstrap and sent to this enclave in order to derive EE_{Key}
Enclave Unseal Key	EU_{Key}	Key generated by the deployment process at the first bootstrap and sent to the enclave, in order to derive EE_{Key}
Enclave Encryption Key	EE_{Key}	Key derived from $EL_{Key} \oplus EE_{Key}$ in order to encrypt $SPMaster_{Key}$
Service Provider Master Key	$SPMaster_{Key}$	Key generated within the enclave on a new team registration in order to encrypt/decrypt $UserSP_{Key}$
User Service Provider Key	$UserSP_{Key}$	Key generated within the enclave when the user is provisioned for SSO authentication, in order to encrypt $Remote_{Key}$
SSO Server Key	$SSOServer_{Key}$	Generated by the server at account creation, in order to encrypt $Remote_{Key}$
Remote Key	$Remote_{Key}$	Generated by the client at account creation, in order to encrypt user's vault
Identity Provider Certificate	IdP_{Cert}	Certificate of public key of the IdP to verify SAML assertion

Table 3: Cryptographic keys and certificates implied in Dashlane-hosted workflows

All keys previously defined is 32 bytes long.

2.4.3   Workflows

Dashlane

KMS
- Generate and encrypt key for enclave at init
- Decrypt key for enclave on reboot
- Verify enclave running code with enclave attestation

Deployment process

Datastore
- Store encrypted data from from the enclave

System initialization

Send enclave local key
Encrypted with an enclave public key

Share enclave unseal key
Secure channel

Data encrypted by/for the enclave
HTTPS

Secure Enclave

Provide cryptographic attestation to prove its runtime environment
At bootstrap, recover the Enclave Encryption Key from Enclave Local Key and Unseal Key (xor)
Encrypt data before requesting for storage into the datastore

Team admin

IdP

Admins

Team creation

Request team creation
Secure channel

DNS challenge
HTTPS

Send IdP certificate
Secure channel

- Verify domain owner with DNS challenge
- Generate Service Provider Master Key for the team
- Signed IdP certificate + domain before storage

Team user

IdP

User authentication

Send SAML assertion signed by IdP
Secure channel

Send User Service Provider Key
Secure channel

- Verify SAML assertion with IdP certificate to authenticate users
- At first user login, generate UserSPkey, and encrypt it with SPMasterkey
- Store encrypted UserSPkey within the datastore
- At further login, recover encrypted UserSPkey from storage, decrypt with SPMasterkey
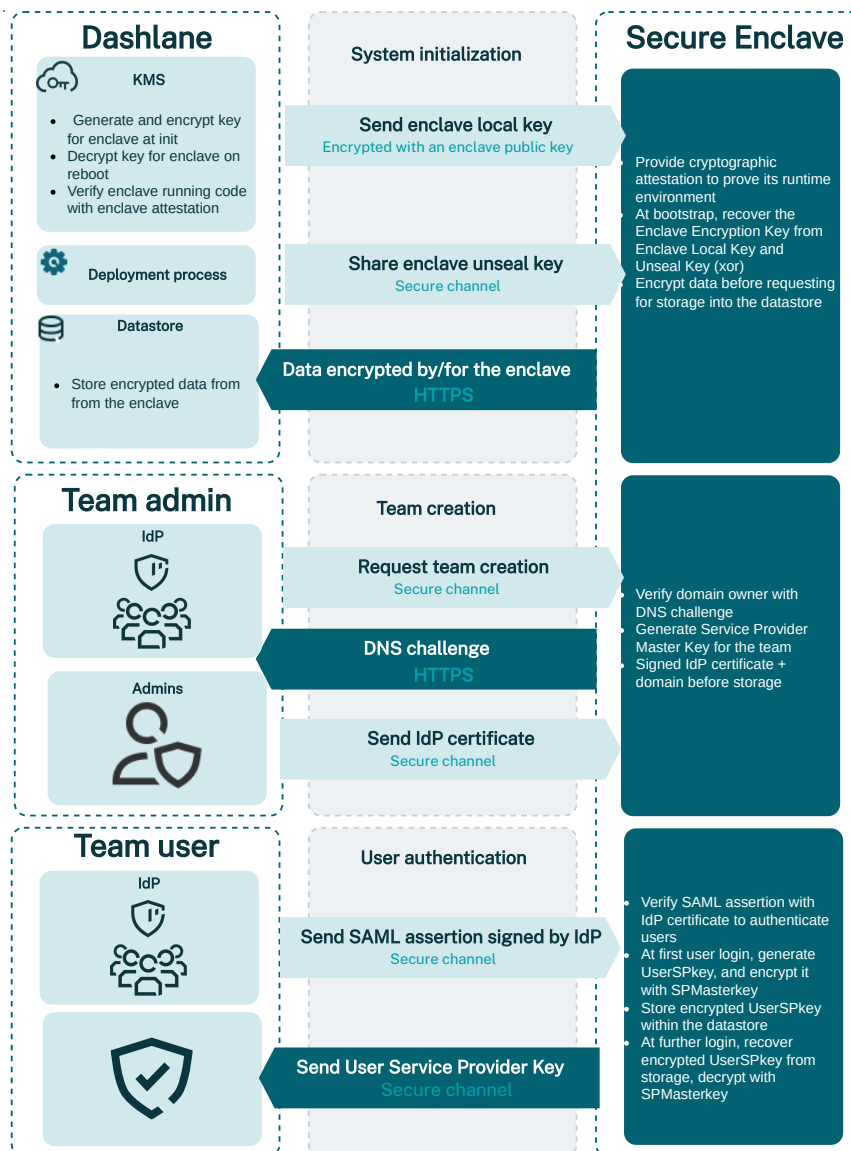
Figure 7: Dashlane-Hosted SSO Workflow

2.4.3.1   Enclave initialization step   The first step is to generate an Enclave Master Key in the KMS and to build access policies to that Enclave Master Key so access is granted only to the enclave. This is done by basing policies on information provided by the attestation of the enclave: when the KMS get a request for the Enclave Master Key, it matches the attestation provided with the policies to grant or deny the request.

Then, the enclave is deployed and requests the KMS to generate an Enclave Local Key and to securely send back to the enclave two versions of the Enclave Local Key: one encrypted by the Enclave Master Key and one encrypted with an ephemeral public key provided by the attestation. The enclave requests the storage of the encrypted Enclave Local Key and keeps the plaintext Enclave Local Key in this volatile memory. This way, if the enclave reboots or a new instance is deployed,

the instance will then request from the storage the encrypted Enclave Local Key then the KMS will decrypt it with the Enclave Master Key. This way, the enclave is provided with the Enclave Local Key to encrypt data, and the Enclave Local Key is never in plaintext outside a secured environment; the enclave or the KMS.

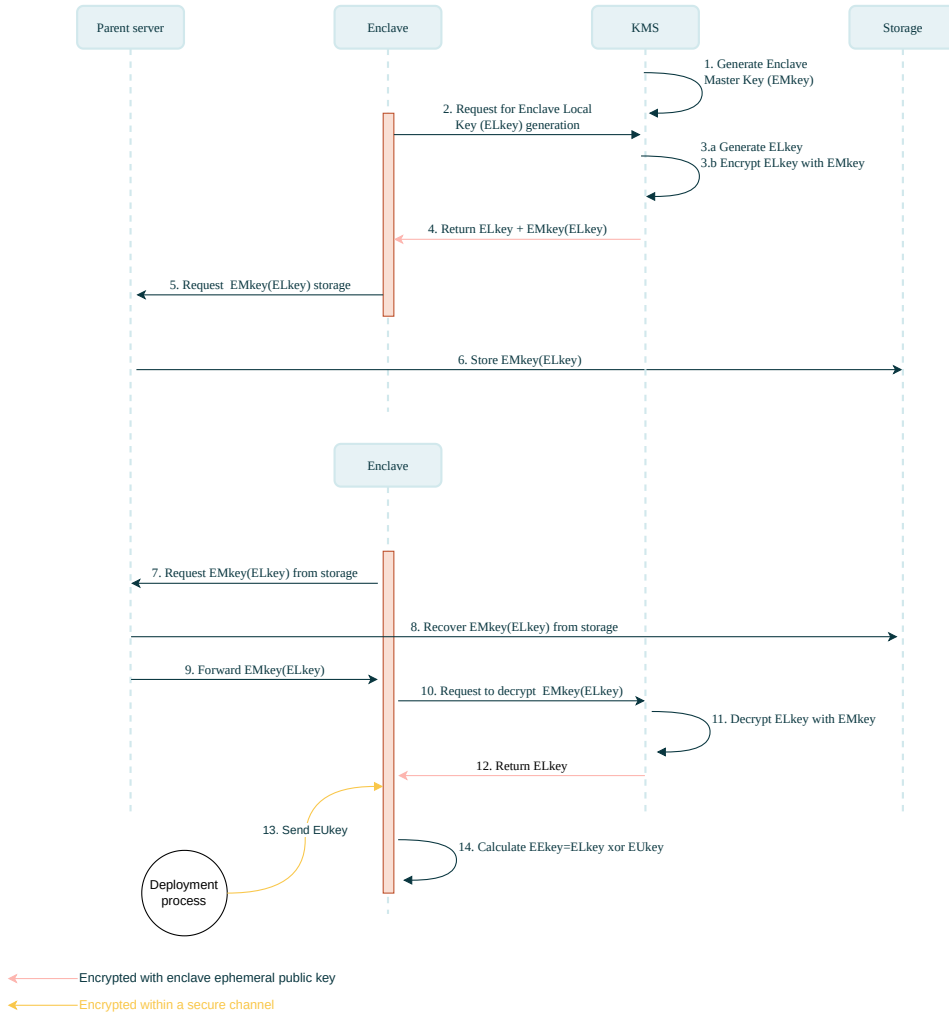Figure 8 describes the workflow to provide secure enclaves with EL_{Key}.

Figure 8: Dashlane Confidential SSO Initialization

Then, the deployment process can mount a secure channel (based on the attestation of the enclave) to send EU_{Key} to the enclave. This way, the secure enclave can derivate the Enclave Encryption Key as follows:

$$EE_{Key} = EL_{Key} \oplus EU_{Key}$$

2.4.3.2   Storage of the secure enclave   A secure enclave is a runtime environment with no persistent storage. Data needs to be encrypted before being passed through the parent server toward the datastore.

Data within the secure enclave requiring persistent storage are the following:

- the Enclave Local Key EL_{Key}, encrypted by the Enclave Master Key EM_{Key}

- Service Provider Master Keys $SPMaster_{Key}$ of each team, encrypted by the Enclave Encryption Key EE_{Key}

- User Service Provider Keys $UserSP_{Key}$ of each user, encrypted by the $SPMaster_{Key}$ of their team

2.4.3.3 Team creation The team creation step is the configuration of the SSO for an organization: the enclave is provided with the IdP certificate to verify SAML assertions for authenticating users of a domain (e.g users with an email from a given domain). The enclave still needs to verify that the admin performing the operation is the owner of the claimed domain: this is to prevent anyone from providing a rogue IdP certificate for a domain they don't own. Indeed, SSO is based on the domain of the email of the user. For example, if a user requests to log in with the username user@example.com, and the domain example.com is linked to an IdP, the user will go through the authentication flow with that IdP. This way, registering an IdP for a domain is a sensitive operation, requiring the secure enclave to perform the domain verification.

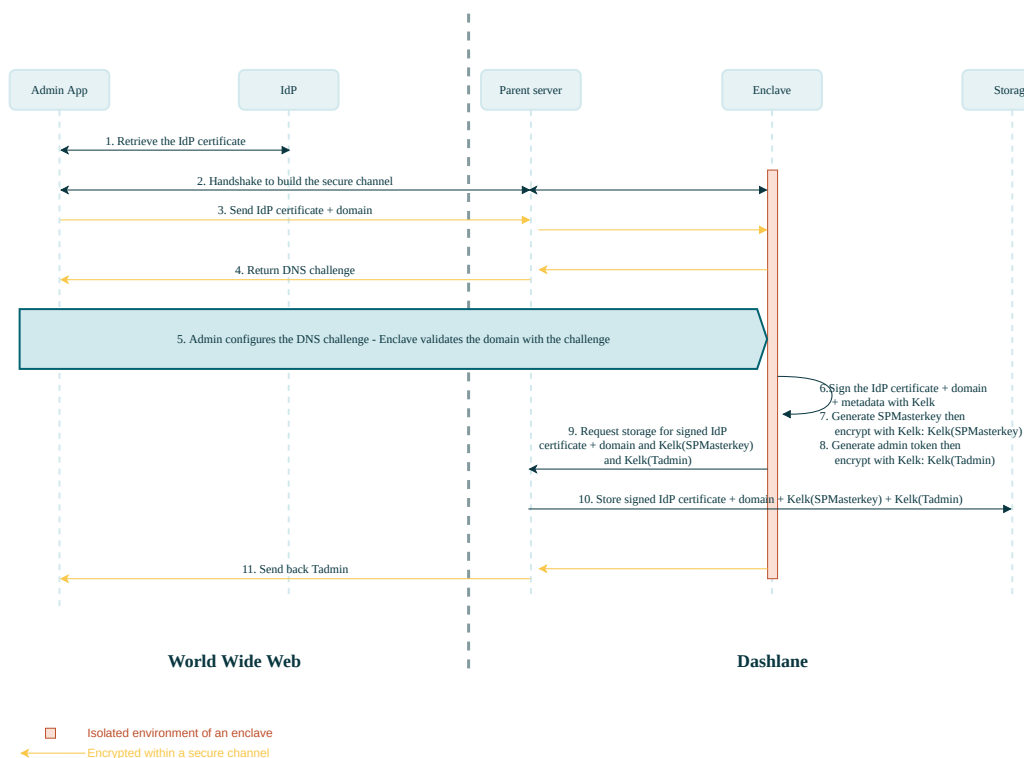The team creation flow is described in Figure 9

Figure 9: Dashlane Confidential SSO Team Creation Flow

1 IT admin of the organization configures the IdP and gets the URL for the IdP endpoint and the IdP certificate of the key, which will sign further users' proof of authentication; IT admin starts the configuration flow of the SSO in the Admin application

2 The Admin application performs a handshake with the enclave to build a se-
cure channel

3 Through the secure channel, the client application sends the IdP certificate
and domain; this is done in the secure channel to protect the IdP certificate's
integrity (to prevent the certificate from being replaced in transit by a rogue
certificate)

4 The enclave sends back a random value to initiate the verification of the do-
main

5 IT Admin and enclave perform the DNS challenge: the goal is to let the enclave
confirm that it is speaking with an owner of the claimed domain; for that, the
IT Admin has to place the random value at the root of the domain and then
the enclave can check this value with the DNS (better with a secured version
of the protocol); this way, the enclave validates that the IT admin is the owner
of the claimed domain

6 The enclave generates a Message Authentication Code (MAC) for the IdP cer-
tificate + domain + metadata from EE_{Key}

7 The enclave generates the Service Provider Master Key for the domain $SPMaster_{Key}$,
then encrypts it with EE_{Key}

8 The enclave generates a token to authenticate admins of the domain (the to-
ken will be shared between admin accounts of the domain), then encrypts it
with EE_{Key}

9 The enclave requests the parent instance to store the signed IdP certificate +
domain, the encrypted $SPMaster_{Key}$, and the encrypted token admin

10 The parent instance stores the signed IdP certificate + domain, the encrypted
$SPMaster_{Key}$, and the encrypted token admin

11 The instance sends back the token admin through the secure channel

2.4.3.4 User SSO login   After the team creation, a user can expect to open their
vault with the SSO flow. Reaching the login page of their client application which
redirects them to the login page of their IdP. After the IdP authenticates the user, it
redirects the user to the client application with a SAML assertion proving their iden-
tity. Then, the client application can send the assertion to the Encryption Service
to receive back $UserSP_{Key}$, decrypting the user's vault.

Until the proof of authentication is sent, the flow is the same for users who per-
form their first login and users who have already enabled their account.

The beginning of the flow is described by Figure 10

From this point, the user is authenticated in the enclave. The flow diverges be-
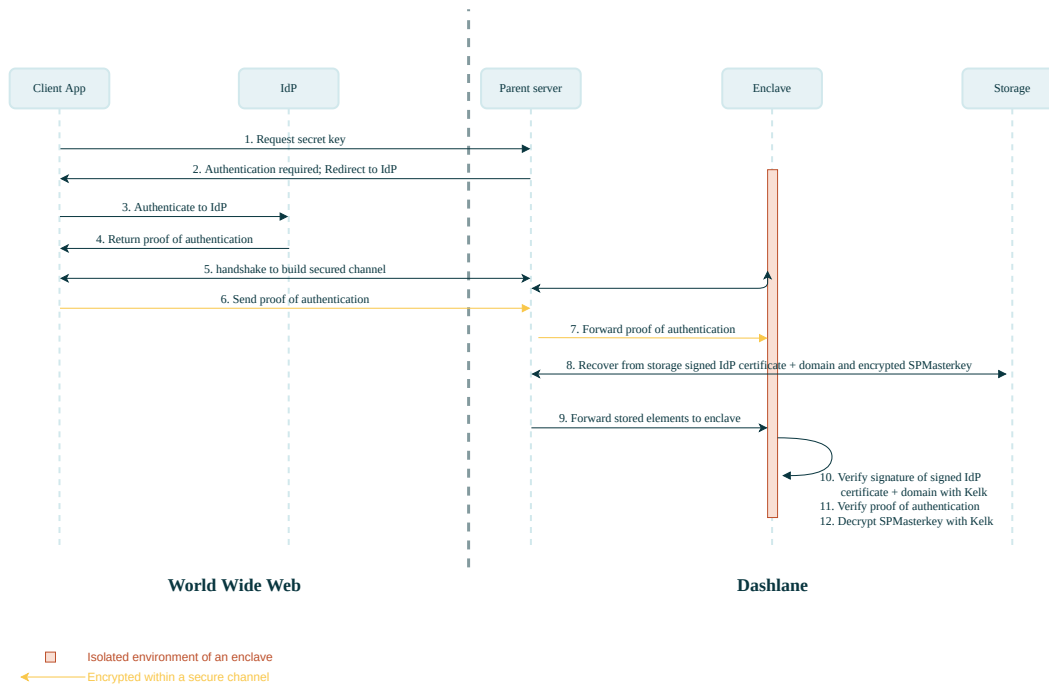tween the first connection account and already enabled account.

Figure 10: Dashlane Confidential SSO - User Login Flow

After users are authenticated (the signature of their proof of authentication is verified) for first connection, the workflow is as described in figure 11a:

13 The enclave generates the $UserSP_{Key}$

14 The enclave encrypts $UserSP_{Key}$ with $SPMaster_{Key}$

15 The enclave requests the parent server store $SPMaster_{Key}(UserSP_{Key})$

16 After confirmation that $SPMaster_{Key}(UserSP_{Key})$ is stored, the enclave sends back to the client application the $UserSP_{Key}$, encrypted in the secure channel

After users are authenticated (the signature of their proof of authentication is verified), for an account already enabled, the workflow is as described in figure 11b:

13 The parent server retrieves $SPMaster_{Key}(UserSP_{Key})$ from the storage

14 The parent server forwards stored elements to the enclave

15 The enclave decrypts the $SPMaster_{Key}(UserSP_{Key})$

16 The enclave sends back to the client application the $UserSP_{Key}$, encrypted in the secure channel
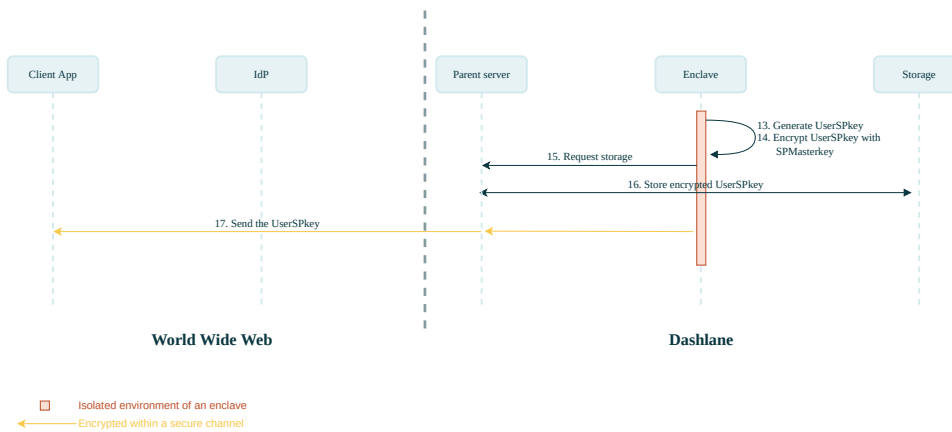
(a) First login
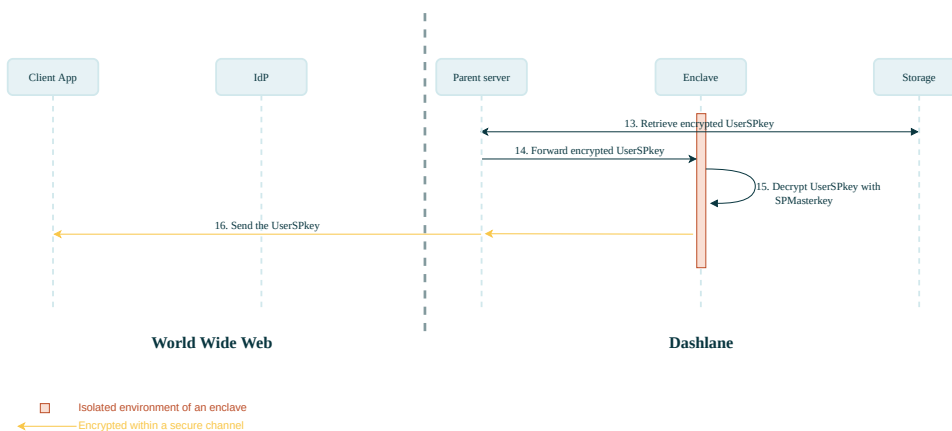
(b) Standard login

Figure 11: Dashlane-Hosted SSO - User Login Flow Part 2

3   Impact on Potential Attack Scenarios

Today, cloud-based services make various choices to encrypt their user data. These choices have important consequences in terms of security.

3.1   Minimal Security Architecture

Cloud services can use a **single private secret**, usually under their control, **to encrypt all user data**. This is obviously a simpler choice from an implementation standpoint, plus it offers the advantage of facilitating **deduplication** of data, which can provide important economic benefits when the user data volume is high. Obviously, this is not an optimal scenario from a security standpoint since if the key is compromised (hacker attack or rogue employee), all user data is exposed.
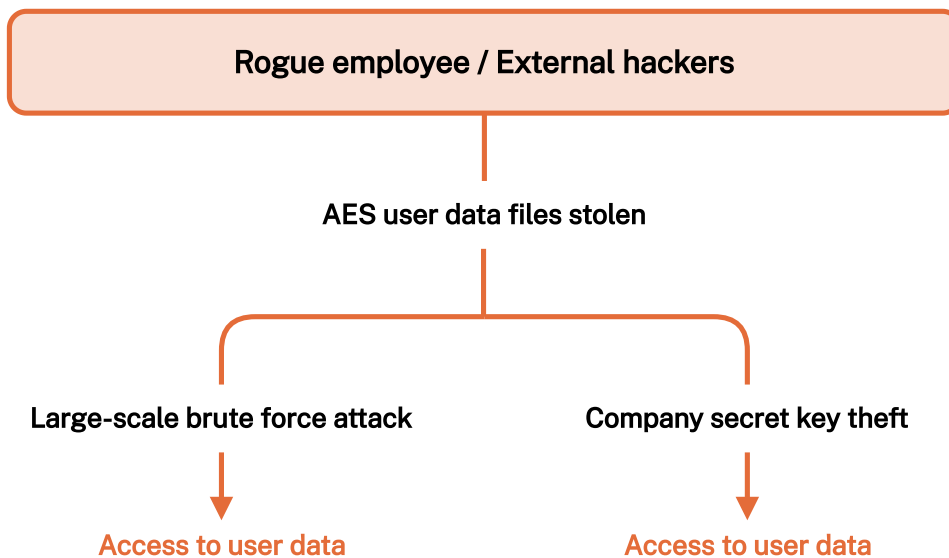
Figure 12: Potential Attack Scenarios With Minimal Security

3.2   Most Common Security Architecture

A better alternative is to use a different key for each user. The most common practice is to ask the user to provide a (strong) $User_{MP}$ and to derive the encryption key for each user from their $User_{MP}$. However, to keep things simple for the user, many services or applications tend to also use the as an authentication key for the connection to their services. This implies that an attacker could access a user's vault by just knowing the Master Password. It could also easily lead to implementation errors (missing salt/rainbow tables attacks, wrong/weak hashing, etc.).
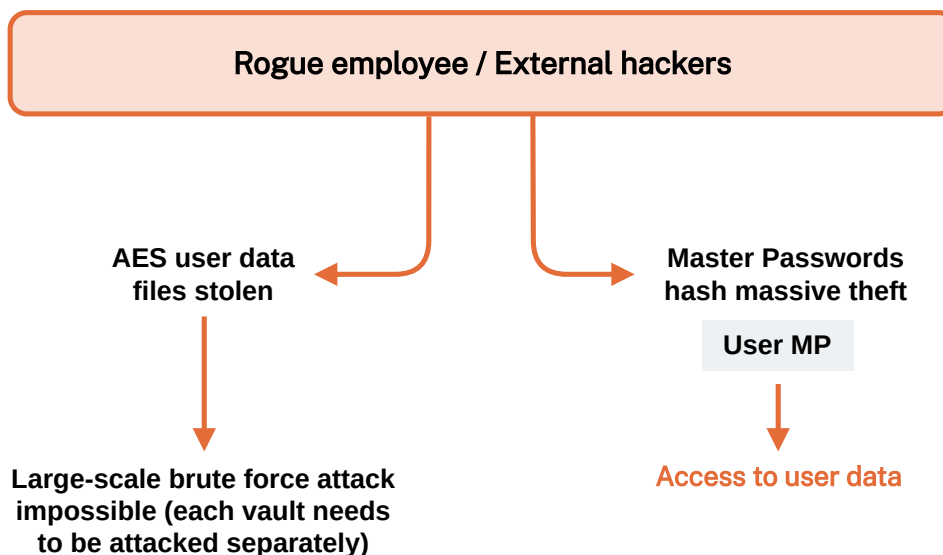

Figure 13: Potential Attack Scenarios With Most Cloud Architecture

3.3   Dashlane Security Architecture

To make this attack scenario impossible, we have made the decision to separate the key used for user data encryption and the key used for server-based authentication (see Figure 14). The user data is encrypted with a key, which is a derivative of $User_{MP}$. A separate $Device_{Key}$ (unique to each device-user couple) is used to perform authentication on Dashlane servers. $Device_{Key}$ is automatically generated by Dashlane. As a result:

- Encryption keys for user data are not stored anywhere.

- No Dashlane employee can ever access user data.

- User data is protected by $User_{MP}$ even if Dashlane's servers are compromised.
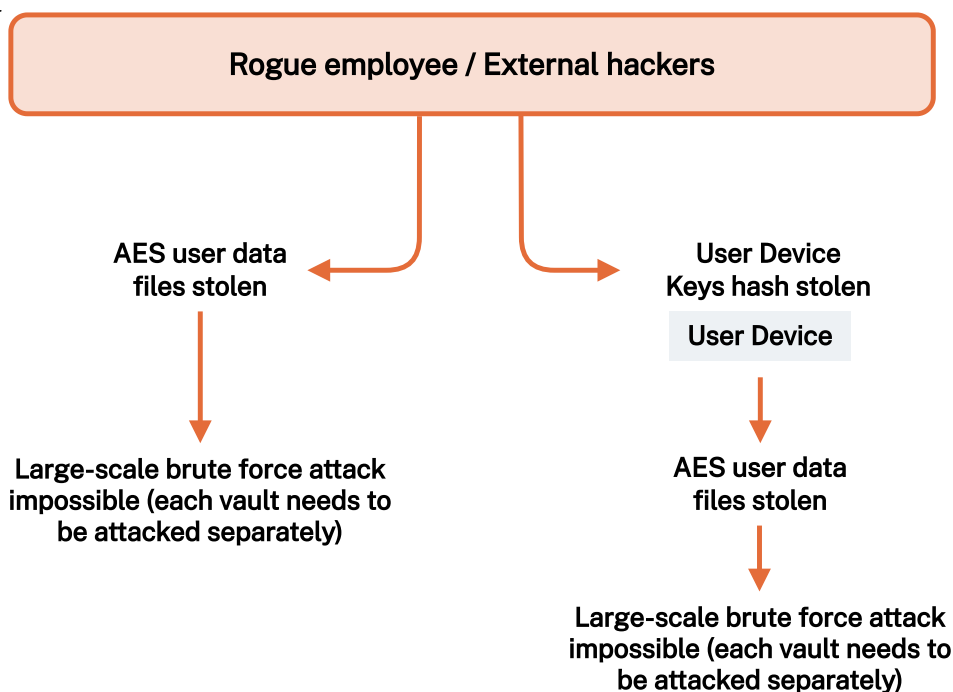

Figure 14: Potential Attack Scenarios With Dashlane's Security Architecture

Even if this scenario happens, a rogue employee or an external hacker would have a very hard time executing a brute force or dictionary attack on the AES user data files, as we use the Argon2d (or PBKDF2-SHA2) algorithm. As the user's data is encrypted using a salted key, which is a derivative of $User_{MP}$, no precomputed attacks should be possible.

3.4   Anti-Clickjacking Provisions

To protect Dashlane users from rogue websites that would attempt to use clickjacking tactics or other JavaScript-based attacks to extract data from the Dashlane application, we have made sure none of the webpage-based interactions involving user data unrelated to this website use JavaScript.

The popups used to trigger form-filling on a webpage use various browser security APIs to prevent control from the JavaScript of the visited page. As a result, a rogue website cannot trigger a click that would cause Dashlane to believe that the user has actually clicked, and therefore, cannot extract information unless the user explicitly clicks in the field.

3.5   Same-Origin Policy

Dashlane automatically logs users into websites. To avoid providing users' information to rogue websites, the same-origin policy is always respected.

First, a credential saved by Dashlane when it has been used on a website with the URL of mysubdomain.mydomain.com will not be automatically filled on another website with the URL of myothersubdomain.mydomain.com. This prevents the credential of a specific website from being provided to another website that share the same top-level domain name.

Also, a credential saved by Dashlane when it has been used on a website with a URL beginning with https will not be automatically filled on another website with a URL beginning with http.

3.6   Memory Protection

A problem can arise if an attacker takes control of the user's client device. In that scenario, the attacker could retrieve the decrypted user data from the memory.

This is an extreme scenario as, in that case, the attacker can take control of many parts, including adding a keylogger to capture $User_{MP}$.

- Mobile operating systems (Android, iOS) ensure that no process can ever access the memory of another process and are not directly affected.

Finally, we believe the system integrity and security between processes is a system function, and Dashlane cannot (and should not) reinvent the wheel and add useless complexity that could lead to other vulnerabilities and have negative side-effects with antivirus software.

For more information on how Dashlane can help you improve password security, please reach out to us and ask.