# 1Password Security Design White Paper

1Password Memberships
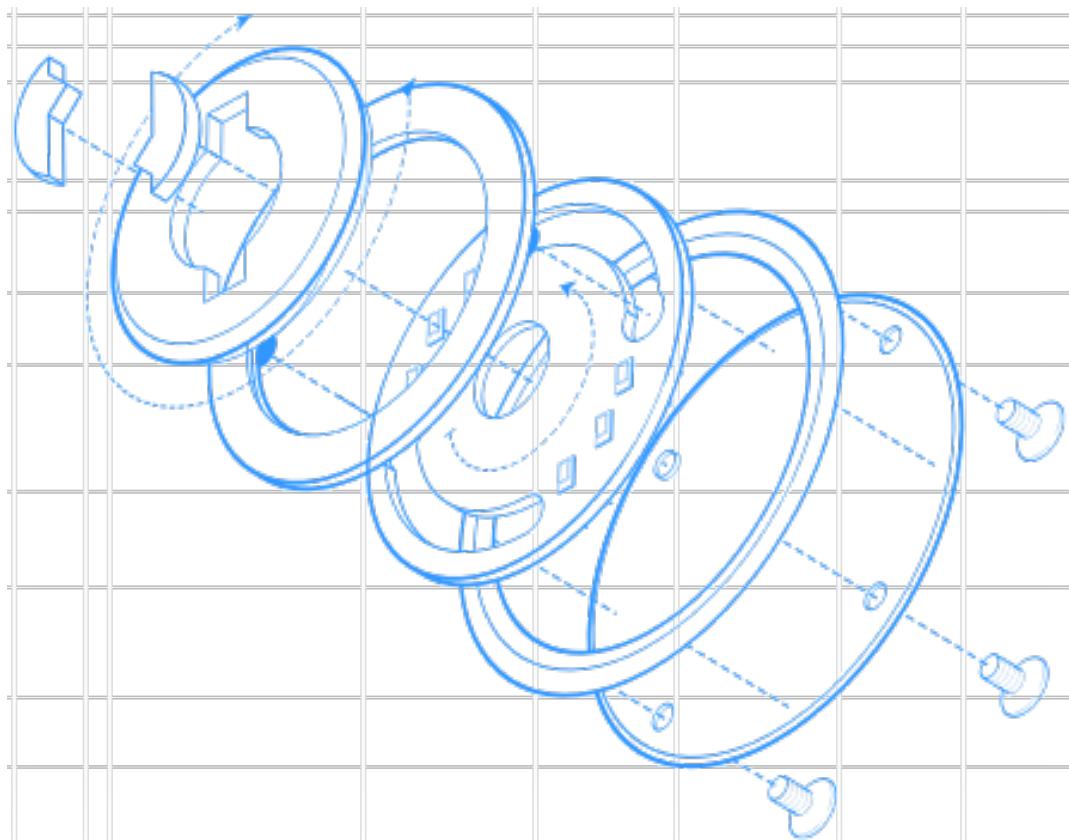
Release 0.5.2

January 22, 2026

Contents

Chapter 1

Key security features

1Password offers a number of notable security features.

True end-to-end encryption All cryptographic keys are generated by the client on your devices, and all encryption is done locally. Details are in "A deeper look at keys."

Server ignorance We're never in the position of learning your account password or cryptographic keys. Details are in "A modern approach to authentication."

Nothing "crackable" is stored A typical web service will store a hash of the user's password. If captured, that can be used in password cracking attempts. Our two-secret key derivation mixes your locally held Secret Key with your account password so data we store cannot be used in cracking attempts. See "Making verifiers uncrackable with 2SKD" for details.

Thrice encrypted in transport When your already encrypted data travels between your device and our servers, it's encrypted and authenticated by Transport Layer Security (TLS) and our own transport encryption. Details are in "Transport security."

You control sharing Only someone who holds the keys to a vault can share that data with someone else. We don't have those keys, so sharing decisions come from you. See "How vaults are shared securely" for details.

Team-managed data recovery We don't have the ability to recover your data if you forget your account password or lose your Secret Key (since you have end-to-end security). But recovery keys can be shared with team members. Details are in "Restoring a user's access to a vault."

Chapter 2

Principles

1Password by AgileBits provides powerful administration of 1Password data (login credentials, for example). This document describes how this happens securely.

The same approach to security that has driven the design of 1Password prior to offering 1Password accounts went into the current design. The first one being we can best protect your secrets by not knowing them.

Principle 1: Privacy by design

It's impossible to lose, use, or abuse data one doesn't possess. Therefore we design systems to reduce the amount of sensitive user data we have or can acquire.

You'll find Principle 1 exemplified throughout our system, from our inability to acquire your account password during authentication through our use of Secure Remote Password (SRP) to our use of two-secret key derivation (2SKD) which ensures we aren't in a position to even attempt to crack your account password. Likewise, our use of end-to-end encryption protects you and your data from us and anyone who may gain access to our servers.

Our second principle follows directly from the first.

Principle 2: Trust the math

Mathematics are more trustworthy than people or software. Therefore, when designing a system, prefer security that is enforced by cryptography rather than software or personnel policy.

Cryptography prevents one person from seeing the items they're not entitled to see. Even if an attacker were able to trick our servers (or people) into misbehaving, the mathematics of cryptography would prevent most attacks. Throughout this document, assume all access control mechanisms are enforced through cryptography unless explicitly stated otherwise.

We also strive to bring the best security architectures to people who are not security experts. This is more than just building a product and system that people want to use, it's part of the security design itself.

Principle 3: People are part of the system

If the security of a system depends on people's behavior, the system must be designed with an understanding of how people behave.

If people are asked to do something that isn't humanly possible, they won't do it. Principle 3 is obvious once it's stated, but sadly it has often been overlooked in the design of security systems. For example, not everyone wants to become a security expert or read this document, but everyone is entitled to security whether or not they seek to understand how it works.

The underlying mechanisms for even seemingly simple tasks and functions of 1Password are often enormously complex. Yet according to Principle 3, we don't wish to confront people with that complexity. Instead, we choose to simplify things so people can focus on accomplishing the task at hand.

Concealing the necessary complexity of the design from users when they just want to get things done is all well and good, but we should never conceal the security design from security experts, system and security administrators, or curious users. Thus we strive to be open about how our system works.

Principle 4: Openness

Be open and explicit about the security design of our systems so others may scrutinize our decisions.

A security system should be subject to public scrutiny. If the security of a system were to depend on aspects of the design being kept secret, those aspects would actually be weaknesses. Expert and external scrutiny is vital[1] both for the initial design and for improving it as needed. This document is part of our ongoing effort to be open about our security design and implementations.

Part of that openness requires that we acknowledge where we haven't (yet) been able to fully comply with all of our design principles. For example, we haven't been able to deny ourselves the knowledge of when you use 1Password in contrast to Principle 1; some finer grained access control features are enforced by server or client policy instead of cryptographically (cf. 2); people still need to put in some effort to learn how to use 1Password properly (cf. 3); and not everything is yet fully documented (cf. 4). We do attempt to be clear about these and other issues as they come up and will collect them in the appendix in "Beware of the leopard."

> **Dangerous bend**
>
> On occations, this document will go into considerable technical detail that may not be of interest to every reader. And so, following the conventions developed in The TEX Book[a], some sections will be marked as a dangerous bend. Most of the text should flow reasonably well if you choose to ignore said sections.
>
> ---
> [a]@TeXBook

Many of those dangerous bend technical discussions involve explaining the rationale for some of the very specific choices we made selecting cryptographic tools and modes of operation. There are excellent cryptographic libraries available, offering strong tools for every developer. But even with the best available tools it's possible to pick the wrong tool for the specific job or to use it incorrectly. We feel it's important not just to follow the advice of professional cryptographers, but to have an understanding of where that advice comes from. That is, it's important to know your tools.

Principle 5: Know your tools

We must understand what security properties a tool has and doesn't have so we use the correct tool the right way for a particular task.

Throughout this document, there will be a process of elaboration. Descriptions presented in earlier sections will be accurate as far as they go, but may leave some details for greater discussion at some other point. Often details of one mechanism make complete sense only in conjunction with details of another that in turn depend on details of the first. And so, when some mechanism or structure is described at some point, it may not be the last word on that mechanism.

---

[1]Goldberg (2013)

Chapter 3

Account password and Secret Key

1Password is designed to help you and your team manage your secrets. But there are some secrets you need to take care of yourself in order to be able to access and decrypt the data managed by 1Password. These are your account password and Secret Key[2] introduced in this section.

Decrypting your data requires all three of the following: your account password, your Secret Key, and a copy of your encrypted data. As discussed below, each of these is protected in different ways, and each individually faces different threats. By requiring all three, your data is protected by a combination of the best of each. Your account password and your Secret Key are your two secrets used in a process we are calling two-secret key derivation (2SKD).

Figure 3.1: *Two-secret key derivation combines multiple secrets when deriving authentication and encryption keys.*
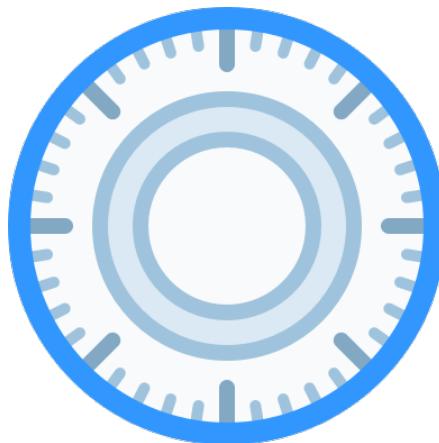
Figure 3.2: *Your account password, like a combination to a lock, is something only you know.*

---

[2]The Secret Key was previously known as the Account Key, and that previous name may appear in internal labelling.

3.1 Account password

One of the secrets used in two-secret key derivation (2SKD) is your account password, and your account password exists only in your memory. This fact is fantastic for security because it makes your account password (pretty much) impossible to steal.

Secrets that must be remembered and used by humans tend to be guessable by automated password guessing systems. We take substantial steps to make things harder for those attempting to guess passwords, but it's impossible to know the capabilities that a well-resourced attacker may be able to bring to bear on password cracking. This is the reason we also include an entirely unguessable secret — your Secret Key — in our key derivation. See Story 1 for an illustration of how your Secret Key comes into play defending you in case of a server breach.

> **Story 1: A (bad) day in the life of your data**
>
> Nobody likes to talk about bad things happening, but sometimes we must.
>
> Oscar somehow gains access to all of the data stored on the 1Password server. We don't know how, and we certainly tried to prevent it, but nonetheless, this is the starting point for our story.
>
> Among the data Oscar acquires is an encrypted copy of your private key. (We store that on our server so that we can deliver it to you when you first set up 1Password on a new device.) If he can decrypt that private key, he'll be able to do some very bad things. Nobody (other than Oscar) wants that to happen.
>
> Oscar will take a look at the encrypted private key and see that it's encrypted with a randomly chosen 256-bit AES key. There's no way he'll ever be able to guess that. But the private key is encrypted with a key derived from your account password (and other stuff) so he figures that if he can guess your account password he will be able to get on with his nefarious business.
>
> But Oscar cannot even begin to launch a password guessing attack. This is because the key that encrypts your private key is derived not only from your account password, but also from your Secret Key. Even if he happens to make a correct guess, he won't know that he has guessed correctly. A correct guess will fail the same way an incorrect guess will fail without the Secret Key.
>
> Oscar has discovered – much to his chagrin and our delight – even all the data held by AgileBits is insufficient to verify a correct guess at someone's account password. "If it weren't for two-secret key derivation, I might have gotten away with it," mutters Oscar. He probably shouldn't have bothered stealing the data in the first place. Without the Secret Keys, it's useless to him.
>
> If Oscar had read this document, he would've known that he can't learn or guess your account password or Secret Key from data held or sent to 1Password.

3.2 Secret Key

Your account password is one of the secrets used in two-secret key derivation (2SKD), and your Secret Key is the other. Your Secret Key is generated on your computer when you first sign up, and is made up of a non-secret version setting, ("A3"), your non-secret Account ID, and a sequence of 26 randomly chosen characters. An example might be A3-ASWWYB-798JRYLJVD4-23DC2-86TVM-H43EB. This is uncrackable, but unlike your account password, isn't something you're expected to memorize or even type on a keyboard regularly.
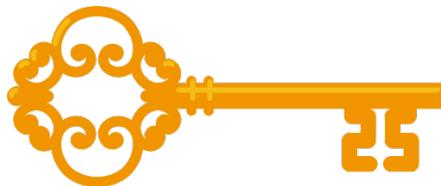
Figure 3.3: *Your Secret Key is a high-entropy secret you have on your devices.*

The hyphens are not part of the actual Secret Key but used to improve readability. The version information is neither random nor secret. The Account ID is random, but not secret, and the remainder is both random and secret. In talking about the security properties of the Secret Key, we'll be talking only about the secret and random portion. At other times we may refer to the whole thing, including the non-secret parts.

There are slightly more than 2^{128} possible Secret Key[3] likely, thus placing them well outside the range of any sort of guessing. But while the Secret Key is unguessable, it's not the kind of thing that can be committed to human memory. Instead of being stored in your head, your Secret Key will be stored on your device by your 1Password client.

3.3 Emergency Kit

AgileBits has no ability to decrypt your or your team's data, nor do we have the ability to recover or reset passwords. The ability to recover or reset the account password or Secret Key would give us (or an attacker who gets into our system) the ability to reset a password to something known to us or the attacker. We therefore deny ourselves this capability.

This means you must not forget or lose the secrets you need to access and decrypt your data. This is the reason we very strongly encourage you to save, print, and secure your Emergency Kit when you first create your account. Story 2 illustrates how it might be used.

Story 2: A day in the life of an Emergency Kit

It's been lonely in this safety deposit box all these months. All I have for company is a Last Will, which does not make for the most cheerful of companions. But I've been entrusted with some of Alice's most important secrets. It's little wonder she keeps me out of sight where I can't reveal them to anyone.

It was a crisp February day that winter before last when Alice first clicked "Save Emergency Kit." She probably thought that she would never need me, but she dutifully (and wisely) printed me out and wrote her account password on me. I already contained her Secret Key along with some non-secret details. She securely deleted any copy of me from her computer and promptly took me to her bank, where I got locked away in this box. Perhaps never to be looked at again.

But today is different. Today I'm the genie released from long imprisonment. Today I'll do magic for my master, Alice. It seems she had a catastrophic disk crash and her backups weren't working as expected. She remembered her account password, but she needed to come to me for her Secret Key. With a fresh copy of 1Password on a new computer, Alice can present the QR code I bear to teach 1Password all the account details, including the Secret Key. All Alice will have to do is type in her account password.

What a day! Now I'm being returned to the bank vault. I hope Alice won't have reason to call upon me again, but we both feel safe knowing I'm here for her.

---

[3]Characters in the Secret Key are drawn uniformly from a set of 31 uppercase letters and digits. With a length of 26, that gives us 31^{26} which is just a tad over 128 bits.

Your Emergency Kit is a piece of paper (once you've printed it) that will contain your account details, including your Secret Key. Figure 3.4 shows an example. It also has space for you to write your account password. If you're uncomfortable keeping both your Secret Key and account password on the same piece of paper, you may decide to store a written backup of your account password separately from your Emergency Kit.
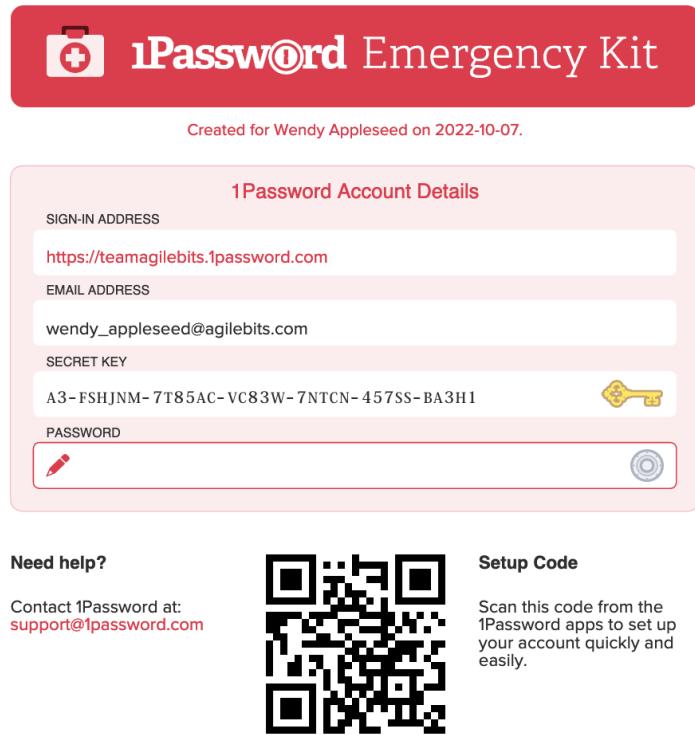
Figure 3.4: *1Password Emergency Kit*

It's a challenge for us to find ways to encourage people to print and save their Emergency Kits. During the 1Password beta period we added a number of places in which we nudged people toward this. This includes making it clearer what should be done with the Emergency Kit, as in Figure 3.5, and by incorporating it among a set of "quests" users are to encourage to complete after first starting to use 1Password.

Figure 3.5: *We encourage users to save their Emergency Kits by a variety of means. One of those means is to make it visually clear what is expected.*

Chapter 4

A modern approach to authentication

1Password is an end-to-end (E2E) encryption system. Thus, the fundamental security of your data rests on the fact that your data is encrypted and decrypted on your local device using keys derived from your secrets which AgileBits has no access to. This end-to-end (E2E) encryption is the primary component of 1Password's security. How that encryption takes place is the subject of "How vault items are secured."

> **End-to-end encryption**
>
> Data is only encrypted or decrypted locally on the users' devices with keys that only the end users possess. This protects the data confidentiality and integrity from compromises during transport or remote storage.

Nonetheless, there is an authentication component to 1Password. For the most part, its job is to ensure an individual only receives the encrypted data they should receive. "Access control enforcement" contains more details about what powers are granted to a client that successfully authenticates.

Traditionally, AgileBits has been wary of authentication-based security, and that wariness has manifested in the design of 1Password in three ways:

1. Our overall design is fundamentally based on end-to-end (E2E) encryption.

2. We've introduced two-secret key derivation (2SKD) to dramatically reduce the risk associated with compromise of authentication verifiers.

3. We don't rely on traditional authentication mechanisms, but instead use Secure Remote Password (SRP) to avoid most of the problems of traditional authentication.

> **Dangerous bend**
>
> Client authentication keys are derived from the same user secrets from which the encryption keys are derived (so you have one password for 1Password). So even though the core of 1Password's security doesn't depend on authentication, we must take extra care in the security of authentication so an attacker can't work backwards from an information leak in an authentication process to discover secrets that might aid in decryption.

4.1   What we want from authentication

Your account password and Secret Key are used to derive a key that's used for authenticating with the service we host.[4] Our service needs to know that you are who you say you are. Your 1Password client needs to prove to the service it has your account password and Secret Key. Only

---

[4]Your account password and Secret Key are also used to derive a different key used for the E2E, which is discussed in later sections.

then will it give you access to your encrypted data (which your client still must decrypt) or other administrative rights.

When Alice authentication to Bob, she does so by proving she has access to a secret[5] that only she should have access to. Bob performs some analysis or computation on the proof that Alice offers to verify it demonstrates Alice's access to her secret.

Story 3: A very traditional authentication exchange

[ALICE approaches castle gate where BOB is on duty as a guard.]
BOB: Who goes there?
ALICE: It is I, Alice. *[ALICE identifies herself.]*
[BOB checks checks his list of people who are authorized to enter the castle to see if ALICE is authorized.]
BOB: What is the password? *[BOB asks ALICE to prove her identity.]*
ALICE: My password is xyzzy. *[She provides proof.]*
[BOB verifies that's the correct password for ALICE.]
BOB: You may enter. *[BOB raises the portcullis and ALICE enters.]*

We want an authentication system to allow Alice to reliably authenticate herself to Bob without introducing other security problems, so there are a number of security properties we would like from a good authentication system. There's substantial overlap among them, but they're all technically separate.

Table 4.1: *Sign-in system desiderata*

Prove client ID	Prove to the server the user holds the user's secret.
Prove server ID	Prove to the user the server holds the server's secret.
Eavesdropper safe	Doesn't reveal any information about either secret in the process.
Not replayable	Can't be replayed by someone who has recorded the process and wants to repeat the exchange to fake a sign-in at another time.
No secrets received	Doesn't reveal any information about the user's password to the server.
Session key	Establishes a new secret that can be used as an encryption key for the session.
No cracking	Server never acquires enough information to facilitate a password-cracking attempt.

4.1.1   Traditional authentication

With a traditional authentication system, the client (such as a web browser) sends a user secret (typically a password) to the server. The server then processes that password to determine whether or not it's correct according to its records. If the server determines it's correct, it will consider that user authenticated.

---

[5]By broadening the definition of "secret," we could also cover biometric authentication with this description.

The simplest way to prove you know a secret is to say it, and that's what the client does in a traditional system. It simply sends the username and the password to the server. A very traditional version of this is illustrated in Story 3.

This traditional design has a number of shortcomings. Indeed, it only satisfies the first authentication disideratum. The most glaring failures of traditional authentication include:

- **Anyone able to eavesdrop on the conversation will learn the client's secret.** In the exchange in Story 3 that would correspond to an eavesdropper hearing and learning Alice's secret password.

- **If the client is talking to the wrong server it reveals its secret to that potentially malicious server.** In Story 3, that would correspond to Bob not really being the castle guard, and Alice revealing to her password to an enemy.

In a typical internet login session, those shortcomings are addressed by Transport Layer Security (TLS) to keep the conversation between the client and the server private as it travels over a network and to prove the identity of the server to the client. As discussed in "Transport security," we make use of TLS but don't want to rely on it.

4.1.2 Password-Authenticated Key Exchange

The modern approach to covering most of the security properties of authentication we seek is to find a way for the client and server to prove to each other they each possess the appropriate secret without either of them revealing any secrets in the process. This is done using a password-authenticated key exchange (PAKE).

Using some mathematical magic, the server and client are able to send each other puzzles that can only be solved with knowledge of the appropriate secrets, but no secrets are transmitted during the exchange. Furthermore, the puzzles are created jointly and uniquely during each session so it's a different puzzle each time. This means that an attacker who records one authentication session will not be able to play that back in an attempt to authenticate.

The "key exchange" part of password-authenticated key exchange (PAKE) establishes a session key: A secret encryption key that the parties can use for the life of the session to encrypt and validate their communication. With 1Password we use this session key to add an additional layer[6] of encryption in our communication between client and server.

A well-designed password-authenticated key exchange (PAKE) – we use Secure Remote Password (SRP), detailed in "Secure Remote Password" – can satisfy all the requirements we've listed except for one. On its own, a PAKE would still leave something crackable on the server, something unacceptable.

4.1.3 Making verifiers uncrackable with 2SKD

A password-authenticated key exchange (PAKE) still doesn't solve the problem of a server acquiring and storing information that could be used in a password cracking attempt. A server holds a long-term verifier that's mathematically related to a long-term authentication secret used by the client. Although this verifier isn't a password hash, it can be considered one for the sake of this immediate discussion. If the client's long-term secret is derived from something guessable (such as a weak password), the verifier stored by the server could be used to help test guesses of that user's password.

We can (and do) take measures to protect the verifiers we store from capture, and the client uses slow hashing techniques in generating the verifier. These are essential steps, but given the nature of what 1Password is designed to protect, we feel those steps are insufficient on their own.

---

[6]This layer provides authenticated encryption for the communication between client and server that is in addition to the security provided by TLS and 1Password's fundamental E2E encryption of user data.

We use two-secret key derivation (2SKD) to ensure data held by the server is not sufficient to launch a password cracking attempt on a user's account password. An attacker who captures server data would need to make guesses at a user's account password **and** have the user's 128-bit strong Secret Key.

It's for this final desideratum we introduced two-secret key derivation (2SKD). With this, the information held on our systems cannot be used to check whether an account password guess is correct or not. Figure 4.1 summarizes which security properties we can achieve with various authentication schemes.

	Traditional	+MFA	PAKE	+2SKD
Prove client ID	✓	✓✓	✓	✓
Prove server ID	✗	✗	✓	✓
No eavesdrop	✗	✗	✓	✓
Prevent replay	✗	?	✓	✓
No secrets received	✗	✗	✓	✓
Session key	✗	✗	✓	✓
No cracking	✗	✗	✗	✓

Figure 4.1: *Authentication schemes and what they do for you. The "+multi-factor authentication (MFA)" column lists the security properties of using traditional authentication with multifactor authentication. The "+2SKD" column lists the security properties of using a PAKE with twosecret key derivation, as done in 1Password. The first column lists our desired security properties.*

Although the internals of our authentication system may appear to be more complex than otherwise needed for a system whose security is built upon end-to-end (E2E) encryption, we need to ensure that an attack on our authentication system doesn't expose anything that could be used to decrypt user data. Therefore the system has been designed to be strong in its own right, and provide no information either to us or an attacker that could threaten the confidentiality of user data.

Chapter 5

How vault items are secured

Items in your vaults are encrypted with Advanced Encryption Standard (AES) using 256-bit keys that are generated by the client on the device, using a cryptographically appropriate random number generator. This generated key becomes your vault key and is used to encrypt and decrypt the items in your vault.

1Password uses Galois Counter Mode (GCM) to provide authenticated encryption, protecting your encrypted data from tampering. Proper use of authenticated encryption offers a defense against a broad range of attacks, including Chosen Ciphertext Attacks (CCA).

$$\text{NewVault(items)}$$

$$K_V \leftarrow\!\!\$\ \text{KGen}()$$

$\textbf{/}$ Add items to V

$$V \leftarrow \varnothing$$

for $x \in$ items

$$y_i \leftarrow \text{Enc}(K_V, x)$$
$$V \leftarrow V \cup \{y_i\}$$

endfor

return V

Figure 5.1: *Algorithm for creating and populating a vault*

The vault key is used to encrypt each item in the vault. Items contain overviews and details that are encrypted separately by the vault key.[7] We encrypt these separately so we can quickly decrypt the information needed to list, sort, and find items without having to decrypt everything in the vault first.

Item overviews include the item fields needed to list items and quickly match items to websites, such as Title, URLs, password strength indicator, and tags. Information that's presented to the user when items are listed, along with the information needed to match an item to a web page (URL), are included in the overview.

Item details include the things that don't need to be used to list or quickly identify them, such as passwords and contents of notes.

---

[7]The overviews and the details are encrypted with the same key. This is a change from the design of the OPVault 1Password data format described in *OPVault Design* (AgileBits 2015).

If you have access to a vault, a copy of the vault key is encrypted with your public key. Only you, the holder of your private key, are able to decrypt that copy of the vault key. Your private key is encrypted with key encryption key (KEK) derived from your account password and Secret Key.

Your private/public key pair is created on your device by your client when you first sign up. Neither we nor a team administrator ever have the opportunity to capture your private key. Your public key, being a public key, is widely shared.

5.1 Key derivation overview

Key derivation is the process that takes your account password and Secret Key and produces the keys you need to decrypt your data and to sign in to the 1Password server. It's described more fully in "Key derivation."

> **Salt**
>
> A cryptographic salt is a non-secret value that is added to either an encryption process or hashing to ensure that the result of the encryption is unique. Salts are typically random and unique.

Your account password will be trimmed and normalized. A non-secret salt is combined with your email address[8] and other non-secret data using hash-based key derivation function (HKDF) to create a new 32-byte salt.

Your account password and the salt are passed to PBKDF2-HMAC-SHA256 with 650,000 iterations. This results in 32 bytes of data, which are combined with the result of processing your Secret Key.

Your Secret Key is combined with your non-secret account ID and the name of the derivation scheme by HKDF to produce 32 bytes of data. This will be XORed with the result of processing your account password.

The resulting 32 bytes of material (derived from both your account password and Secret Key) are your Account Unlock Key (AUK) which is used to encrypt the key (your private key) that's used to decrypt the keys (vault keys) that are used to encrypt your data.

By encrypting copies of vault keys with an individual's public key, it becomes easy to securely add an individual to a vault. This secure sharing of the vault key allows us to securely share items between users.

5.2 A first look at key sets

We organize the public/private key pairs together with the symmetric key that's used to encrypt the private key into key sets. Our key set make extensive use of JSON Web Key (JWK) objects.

---

[8]The reasons for binding your encryption key tightly with your email address are discussed in "Restoring a user's access to a vault."

```
"uuid": string,           // Unique ID of this item (hexadecimal)
"encSymmetricKey": JWK,    // encrypted key that encrypts private key
"encryptedBy": string,     // UUID of key that encrypts encSymmetricKey
"publicKey": JWK,          // public part of key pair
"encPrivateKey": JWK,      // private part, encrypted
```

Figure 5.2: *A key set is a collection of JWK keys together with an identifier and information about what other key set is used to encrypt it.*

A common type of key set will have the structure listed in Figure 5.2. When we speak of encrypting a key set, we generally mean encrypting the symmetric key that's used to encrypt the private key.

key set are fairly high-level abstractions; the actual keys within them have a finer structure that includes the specifications for the algorithms, such as initialization vectors. Symmetric encryption is AES-256-GCM, and public key encryption is RSA-OAEP with 2048-bit moduli and a public exponent of 65537.

Story 4: A day in the life of an item being created

In the beginning there was the vault, but it was empty and had no key.

And Alice's 1Password client called out to the cryptographically secure random number generator, "Oh, give me 32 bytes," and there were 32 random bytes. These 256 bits were called the "vault key."

And the vault key was encrypted with Alice's public key, so only she could use it; a copy of the vault key was encrypted with the public key of the Recovery Group, lest Alice become forgetful.

Alice went forth and named things to become part of her vault. She called forth the PIN from her account on the photocopier and added it to her vault. The photocopier PIN, both its details and its overview, were encrypted with the vault key.

And she added other items, each of its own kind, be they Logins, Notes, Software Licenses, or Credit Cards. And she added all of these to her vault, and they were all encrypted with her vault key. On the Seventh day, she signed out.

And when she signed in again, she used her account password, and 1Password used her Secret Key, and together they could decrypt her private key. With her private key she decrypted the vault key. And with that vault key she knew her items.

And Alice became more than the creators of 1Password, for she had knowledge of keys and items which the creators did not. And it was good.

Once a vault has been created, it can be securely shared by encrypting the vault key with the recipient's public key.

5.2.1 Flexible, yet firm

Since the right choices for the finer details of the encryption schemes we use today may not be the right choices tomorrow, we need some flexibility in determining what to use. Therefore, embedded within the key set are indications of the ciphers used. This would allow us to move from RSA with 2048-bit keys to 3072-bit keys, relatively easily when the time comes, or to switch to Ellipic Curve Cryptography (ECC) at some point.

Because we supply all the clients, we can manage upgrades without enormous difficulty.

> **Story 5: Days in the life of an algorithm upgrade**
>
> ***Setting*** Some time in the not-so-distant future.
> ***Day one*** "Hmm," says Patty. "It looks like 2048-bit RSA keys will have to be phased out. Time we start transitioning to 3072-bit keys."
> ***The next day*** We ensure all our clients are able to use 3072-bit keys if presented with them.
> ***Some weeks later*** We release clients that use 3072 bits when creating new public keys. (Public keys are created only when a new account is created or a new Group is created within a team.)
> ***Further along*** "We should go further and start replacing the older keys." (Of course, we can't replace anyone's keys, as we don't have them.)
> ***After some development*** We issue updated clients that generate new public keys, and anything encrypted with the old key will be re-encrypted by the client with the new key.
> ***Time to get tough*** We can have the server refuse to accept new data encrypted with the older keys. The server may not have the keys to decrypt these key sets, but it knows what encryption scheme was used.
> ***More bad news on 2048-bit keys*** We learn that even decrypting stuff already encrypted with the older keys turns out to be dangerous. [**Editor's Note:** This is a *fictional* story.] We need to prevent items encrypted with 2048-bit keys from being trusted automatically.
> ***Drastic measures*** If necessary, we can issue clients that will refuse to trust anything encrypted with the older keys.

Building in the flexibility to add new cryptographic algorithms while limiting the scope of downgrade attacks isn't easy. But as illustrated in Story 5, we're the single entity responsible for issuing clients and managing the server, so we can defend against downgrade attacks through a combination of client and server policy.

Chapter 6

How vaults are shared securely

Sharing items among members of the same 1Password account happens at the vault level. This allows those members to share and mutually maintain sets of items. Through the magic of public key encryption, this happens without the 1Password service (or us, its operators) ever having the keys or secrets necessary to decrypt shared data.

As described in "How vault items are secured," each user has a personal key set that includes a public/private key pair, and each vault has its own key used to encrypt the items within that vault. At the simplest level, to share the items in a vault, one merely needs to encrypt the items with the public key of the recipient.

> **Story 6: A day in the life of a shared vault**
>
> Alice is running a small company and would like to share the office Wi-Fi password and the keypad code for the front door with the entire team. Alice will create the items in the Everyone vault, using the 1Password client on her device. These items will be encrypted with the vault key. When Alice adds Bob to the Everyone vault, Alice's 1Password client will encrypt a copy of the vault key with Bob's public key.
>
> Bob will be notified that he has access to a new vault, and his client will download the encrypted vault key, along with the encrypted vault items. Bob's client can decrypt the vault key using Bob's private key, giving him access to the items within the vault and allowing Bob to see the Wi-Fi password and the front door keypad code.
>
> The 1Password server never has a copy of the decrypted vault key, and is never in a position to share it. Only someone with that key can encrypt a copy of it. Thus, an attack on our server could not result in unwanted sharing.

6.1   Getting the message (to the right people)

It's important that the person sharing a vault shares it with the right person, and uses the public key of the intended recipient. One of the primary roles of the 1Password server is to ensure that public keys belong to the right email addresses.

> **Dangerous bend**
>
> 1Password does not attempt to verify the identity of an individual. The focus is on tying a public key to an email address. Internally we bind a key set to an email address, but we have no information about who controls that email address.

Connecting users with their keys as they register, enroll new devices, or simply sign in is a fundamental part of the service. How this happens without giving us the ability to acquire user secrets is the subject of the next section.

Chapter 7

How items are shared with anyone

As described in "How vaults are shared securely," sharing items among members of the same 1Password account happens at the vault level But it's also possible to securely share a copy of individual items with individuals who are not members of the same 1Password account or even 1Password users at all. The mechanism, however, is entirely different, largely because the recipient can't be expected to authentication in the same way as a 1Password user would, and the recipient doesn't have a public key the sender can use.

The 1Password item sharing service allows a 1Password user to make a copy of an item available to anyone, whether the recipient is a 1Password user or not. In this section we'll often follow user documentation and the user interface in calling this "sharing," but we we'll also write in terms of sending and receiving to help accentuate the distinction between this mechanism and the sharing of vaults.

7.1   Overview

There are six components to the overall picture.

1Password client The 1Pasword application with which users directly interact. This includes not only applications such as the 1Password iOS app, but also the web client and browser extensions acting as a client. We we'll sometimes refer to this as "the sender's client."

1Password service The principle 1Password service.

Item sharing servive The share service which, among other things, will store the encrypted shared copies.

Item sharing client The share web client. It operates in the recipient's browser and is downloaded from the item sharing service. We'll sometimes refer to this as "the recipient's client."

Sender The human interacting with their 1Password client.

Recipient The human interacting with the item sharing client.

Figure 7.1 illustrates which components talk to which. In particular, the 1Password client communicates only with the sending user and 1Password service. The 1Password server communicates with the item sharing service. Item sharing client communicates only with the receiving user and the item sharing service. And the sender will communicate directly with the recipient outside of 1Password using a communication channel of their choice. Additionally, the figure shows the encrypted copy of the item and key, and that their decryptions follow different paths.
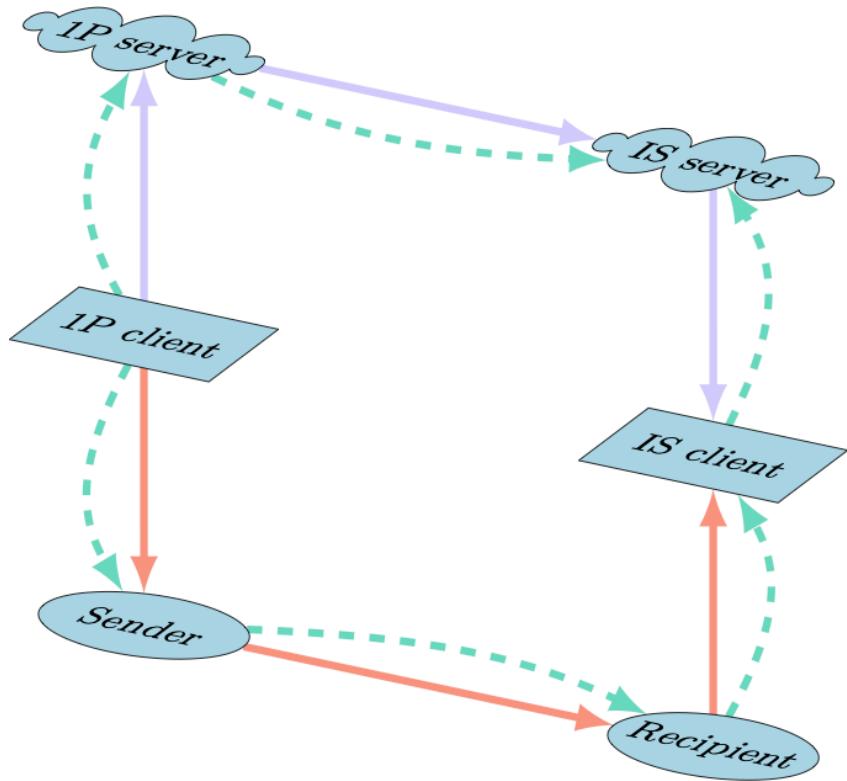
Figure 7.1: *Item sharing, key, and token flow. The solid purple arrows show the flow of the encrypted item, solid red arrows show the flow of the encryption key, and dashed green arrows show the flow to the retrieval token. "IS Server" and "IS Client" refer to the item sharing server and client. Item encryption and decryption can only occur where the key and the item are together, which is at the clients. The share token is used for authenticating a download of the encrypted item by the item sharing server.The 1Password and item sharing servers never have access to the share key. This figure doesn't include flow of audit and share status information.*

7.2 1Password client

The sender's client needs to do two things: create a share link and upload an encrypted copy of the shared item to the 1Password service. The sender also needs to transmit the share link to the recipient independently of 1Password services.

7.2.1 Making a share link

The share link looks something like what is shown in Figure 7.2.

Share link

```
https://share.1password.com/s#base64-encoded-secret
```

Figure 7.2: *A share link is a URL that contains in its fragment information required to identify the shared item, along with the key necessary to decrypt it.*

At a high level creating the share link involves five steps performed by the sender's client.

1. Obtain the user item to be shared and decrypt it. The item must be something the sender

can read and decrypt, and the sender must also have other appropriate permissions for the vault.

2. Create a public unique identifier, a retrieval token, and an encryption key for this share.

3. Encrypt a copy of the item with the share key.

4. Upload the encrypted data to the share service along with the identifier and sender chosen access options.

5. Present the sender with the share link.

As always, the key generation, encryption, and creation of the share link are performed entirely on the sender's client. The 1Password service never has access to the decryption keys nor the decrypted item.

7.2.2 Client's first steps

The requirements of step 1 involve a number of mechanisms. The requirement that the sender be able to decrypt the item is cryptographically enforced, as they would never be able to re-encrypt anything they cannot decrypt. Additionally, the client will only present the user the option to share if the sender has the appropriate vault permission (detailed in the description of the upload step).
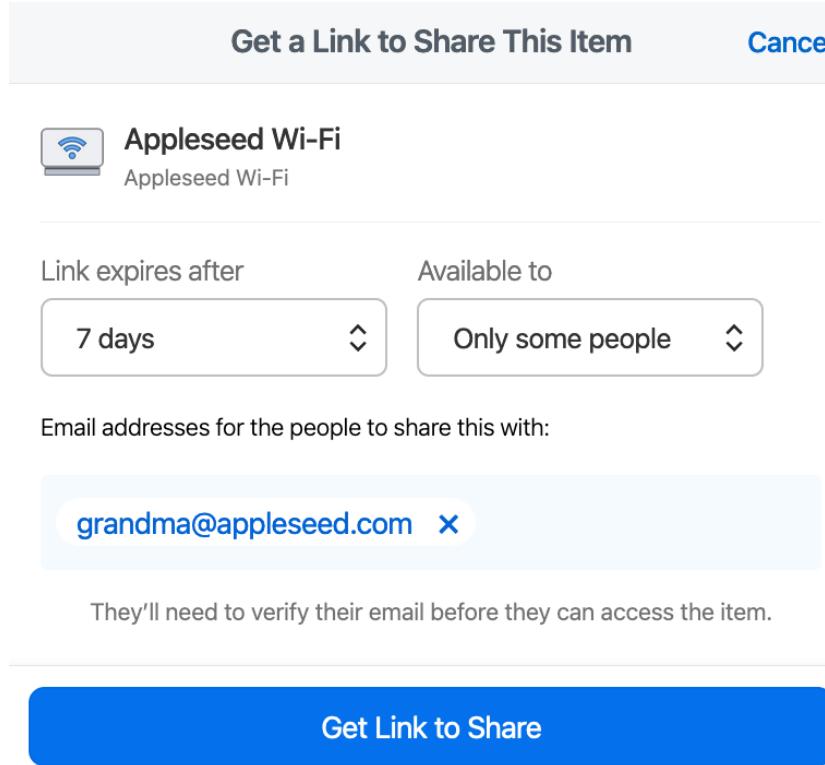
Figure 7.3: *1Password client gets information from sender*

The sender is presented with certain configuration options including whether recipients need to demonstrate control of certain email addresses, the required email addresses if any, the amount of time the shared item should be available, and how many times the recipient can retrieve the item. These options will play a role in the retrieval process. The client will check whether the options are consistent with account policy at this time.

7.2.3 Encryption and key generation

In step 2, the client generates 32 random bytes to be the share secret. The 32-byte encryption key, the 16-byte public Universally Unique Identifier (UUID), and the 16-byte retrieval token are derived from the share secret using hash-based key derivation function (HKDF) as detailed in Figure 7.4.

$$s \leftarrow\!\!\$ \ \{0,1\}^{256} \quad \text{\textbf{/}} \text{ share secret, } s, \text{ is 256 random bits}$$

$\text{\textbf{/}}$ Derive share key k from share secret

$$i_k \leftarrow \mathsf{UTF8Bytes}(\texttt{"share\_item\_encryption\_key"}) \quad \text{\textbf{/}} \text{ Info bytes}$$

$$k \leftarrow \mathsf{HKDF}_{\mathsf{SHA256}}(\mathsf{key} = s, \mathsf{salt} = 0, \mathsf{info} = i_k, \mathsf{size} = 32)$$

$\text{\textbf{/}}$ Derive a public UUID u from share secret

$$i_u \leftarrow \mathsf{UTF8Bytes}(\texttt{"share\_item\_uuid"}) \quad \text{\textbf{/}} \text{ Info bytes}$$

$$u \leftarrow \mathsf{HKDF}_{\mathsf{SHA256}}(\mathsf{key} = s, \mathsf{salt} = 0, \mathsf{info} = i_u, \mathsf{size} = 16)$$

$\text{\textbf{/}}$ Derive retrieval token t from share secret

$$i_t \leftarrow \mathsf{UTF8Bytes}(\texttt{"share\_item\_token"}) \quad \text{\textbf{/}} \text{ Info bytes}$$

$$t \leftarrow \mathsf{HKDF}_{\mathsf{SHA256}}(\mathsf{key} = s, \mathsf{salt} = 0, \mathsf{info} = i_t, \mathsf{size} = 16)$$

Figure 7.4: *Creating shared IDs and secrets. The client generates a share secret, from which it derives an encryption key, a public UUID, and a retrieval token.*

The first part of step 3 involves making a copy of the item to be shared. The copy is identical to the one remaining in the vault except that attachments and password history are removed. The existence of attachments and password history in an item may not always be salient to the sender, and so those are excluded from the copy. The copy is then encrypted using the identical methods used for encrypting items in vaults.

7.2.4 Uploading the share

In step 4, the client uploads the encrypted item along with the public Universally Unique Identifier (UUID) and the retrieval token to the 1Password service. The service will reject the request unless sending items is allowed by account policy; the recipient specification is consistent with account policy; and the sender has read, reveal password, and send item permissions for the vault containing the original item. The share secret, which contains the the share key, is never passed to the service. Additionally, the UUID of the vault and specific item are uploaded, along with the version number of the vault. This metadata allows the user and vault administrators to manage shares. Account policies can specify whether share retrievals must be tied to recipient email addresses and whether those are limited to specific email domains.

The configuration options selected by the sender are also part of this upload. The server will check whether they're consistent with account policy (whether retrieval is restricted to people with particular email addresses) and whether the email addresses conform to that policy. Additionally, the server will ensure the availability time requested by the client doesn't exceed server or account policy. Upon success, the server responds with the non-fragment part of the share link. In current configurations, this is `https://share.1password.com/s`.

After a successful upload, the client will present the user with the share link (step 5).
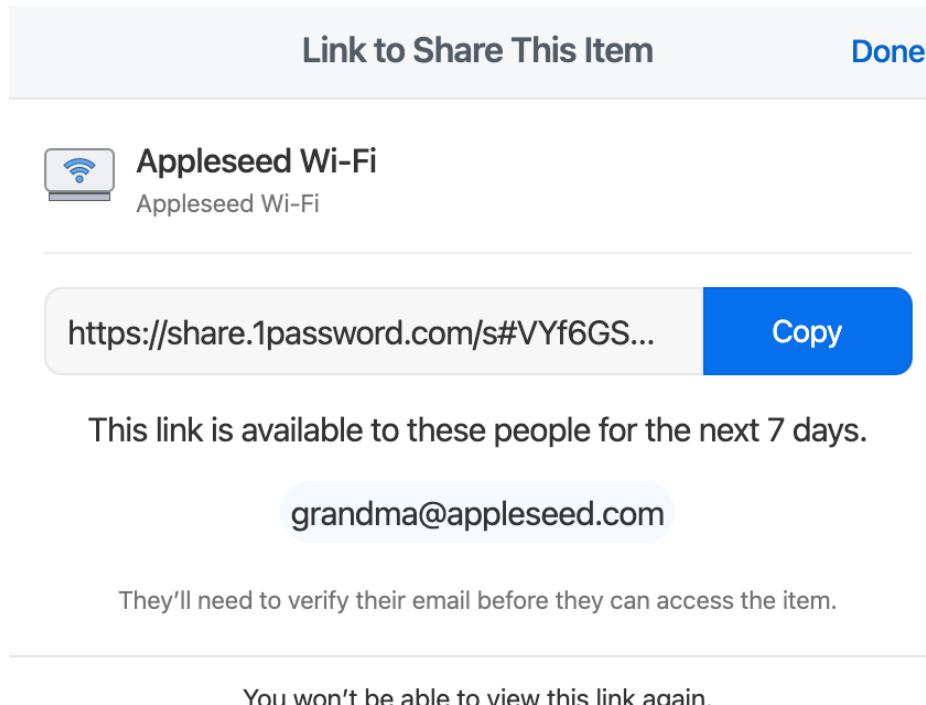
Figure 7.5: *1Password client presents item sharing share link.*

The client appends the share secret (base64 encoded) as a fragment to the URL returned by the server. The fragment in the share link contains the share secret from which the retrieval token and encryption key are derived, and therefore must not be transmitted to intended recipients via the 1Password service.

> **Dangerous bend**
>
> The share link places the identifier and the decryption key within a URL fragment. In general fragments are never transmitted and are used solely by the clients as additional information on handling the retrieved resource. By putting the share secret in the fragment, we not only prevent ways in which the secret could be exposed, but we also make it clear this is a purely local secret.

> **Discussion 1: Fragment abuse?**
>
> At a superficial level, it may appear that placing the item identifier within a fragment is a minor abuse of the standards defining URLs. The non-fragment part of a URL is supposed to fully identify the resource to be retrieved, while that portion of our share links only identifies the 1Password share service. Fragments, however, are also intended to provide information to the client about handling or further processing the retrieved resource, and this is exactly how we use the fragment. Thus our fragment, even with its inclusion of a resource identifier, may be closer to the spirit of the standards than would be including it within the path or a query string. In any event, we don't anticipate the IETF to send an enforcement squad to our door for our use of fragments.

7.2.5   Sharing the share link

The sender needs to communicate the share link to the recipient, and this has to happen out of band. That is, the unencrypted share link – with its secrets – must never be made available to either the 1Password or item sharing services. This ensures the servers never have access to both encrypted user data and the keys needed to decrypt them.

This leaves it to the sender to choose how to get the share link to the intended recipients. The 1Password client may provide as a convenience a way to launch a variety of sharing mechanisms, such as email or messaging tool, on the sender's system.

7.3   Server to server

The 1Password service is responsible for passing the share to the item sharing service. The 1Password service communicates with the item sharing service using our inter-service communication architecture;[9] the item sharing service is hosted at `share.1password.com` and is a separate service despite being in a subdomain of `1password.com`.

> **Dangerous bend**
>
> The item sharing server itself is hosted in the European Union, despite the .com top-level domain. This same EU-based service is used by all 1Password servers, including those in the United States and Canada. This way, recipient email addresses – as they are passed from 1Password server to item sharing server – never leave the European Union.

The 1Passsword server passes to the item sharing service what it received from the sender's client, along with information about the authentication user and account, it adds its own timestamp, and the number of seconds the item is to be available to the current time to create an expiry time.

7.3.1   Audit and status queries

The 1Passsword server has the ability to query the item sharing service about the state of existing shares. The queries identify items by their account, vault, and item Universally Unique Identifier (UUID). As discussed in far too much detail in Discussion 2, UUIDs are never expected to be secret, so guaranteeing the requests are authorized cannot depend on knowledge of those UUIDs. To ensure only authorized individuals are able to see the status or audit events regarding a share:

- These queries are performed over the same mutually authenticated channel (not yet documented).

- The 1Passsword server ensures the account identifier in the query is honest.

- The 1Passsword server has the responsibility to ensure the authenticated user creating the request is authorized to make such requests for that account.

7.4   Share pickup

When the recipient follows a share link, their browser will fetch the page at `https://share.1password.com/s`. The browser won't transmit the fragment containing the share secret. The page contains a item sharing web client, software that will run within the user's browser on their own device. The item sharing client running within the user's browser is, however, able to see and make use of the fragment portion of the share link, and thus will have the share secret.

---

[9]As yet undocumented, but it involves mutual authentication independent of TLS.

The client uses the share secret as detailed in Figure 7.4 to derive three things:

Share key The share key is the symmetric key that was used to encrypt the item and is necessary to decrypt it. This key is never made available to any 1Passsword service.

Share token The share retrieval token is a secret that is shared between the user and the share service.

Share UUID The share Universally Unique Identifier (UUID) is a non-secret record locator used by the share service to locate the item in its data store.

The item sharing client will send a request to the share service requesting the share by UUID. The share token is passed to the item sharing server in an HTTP header `OP-Share-Token`, as is the norm for bearer tokens. Neither the share secret nor the share key are ever sent to the service.

Discussion 2: Knowledge of identifiers should never prove anything

There's some duplication between the UUID and the share token. The share token is sufficient to uniquely identify the share, and knowledge of the UUID offers the same proof of receipt of the share link as knowledge of the share token does. The fact that we don't combine both of those functions into a single value provides an opportunity to explain a design principle throughout 1Passsword.

In the United States, Social Security Numbers (SSNs) were never designed to be secrets they were unique identifiers. But in the second half of the 20th century, banks offering services by telephone began to take knowledge of a caller's SSN as proof of identity. Most systems continued to treat SSNs as non-secrets for a very long time, and changing those systems has proved to be enormously difficult. Credit card numbers followed a similar story with the advent of telephone shopping.

Co-opting knowledge of account or personal identifiers for authentication must have seemed like an easy strategy at the time. We're still plagued with the consequences half a century later.

Throughout the design of 1Passsword we have insisted that knowledge of a UUID must never be used as an authentication mechanism. This not only gives us the freedom to design protocols in which UUIDs never have to be kept secret, it also means we don't have to worry about future uses. At the moment, share tokens and share UUIDs pass through the same hands and over the same channels, but there may be a time when we use or log UUIDs in ways that would be inappropriate for share tokens or require new security properties of share tokens. By holding ourselves to our design pattern now, we prevent a substantial category of security bug in the future.

The item sharing service will first check the validity and existence of the Universally Unique Identifier (UUID). A malformed or unknown UUID will result in an error response to the item sharing client. If the UUID is valid, it then compares[10] the share token with what it has stored. If no further authentication is required, it returns the encrypted item to the item sharing client, which can then use the share key to decrypt the item and render it for the recipient.

The item sharing server creates and stores an audit event record for a successful access. The audit record includes identifiers for the share and the accessor as well as the IP address and HTTP user agent of the and item sharing client. For business accounts, audit events are available to the mangers of the account from which the item was sent.

7.4.1 Additional authentication

In all cases, the item sharing client needs to provide the server with the share token and will need have the share key to be able to decrypt the share, but there may be additional authenti-

---

[10]All comparisons of secrets, including this one, are performed using constant time comparison methods.

cation requirements. At the present time, additional authentication is based on control of email addresses.
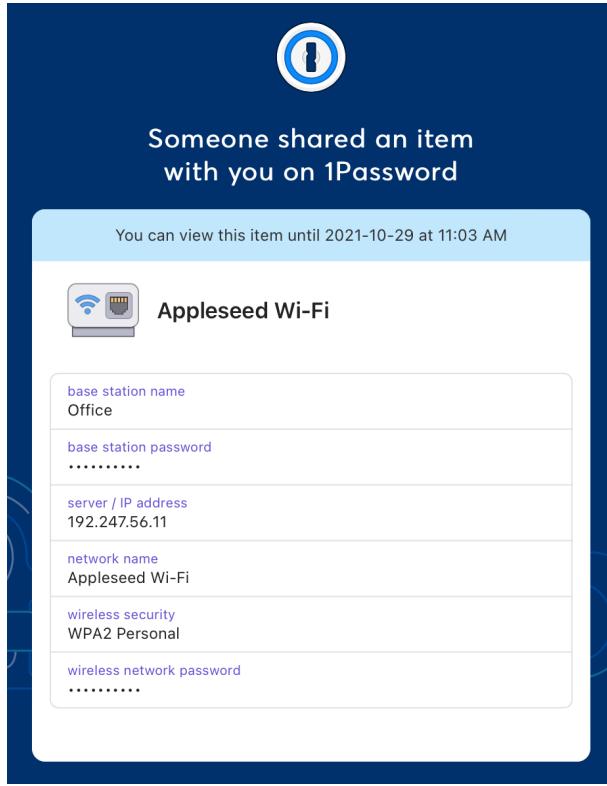
Figure 7.6: *Item sharing client item view.*

In the case that email authentication is required, the share recipient (or their client) needs to either prove they can read email sent to the specified email address or prove it has previously offered such proof. In the first case, the share recipient is offered to have a verification email sent to the address associated with the share by the sender. The email will contain a randomly chosen six-digit verifier the user can enter into the item sharing client, which will send the item to the item sharing server.

After successful email verification, the server will issue an accessor token to the client, which the client will write to its local storage. For future share retrievals, the client can provide this access token to show that it has previously succeeded with email verification.

7.4.2 Client analytics

The item sharing client may offer its user buttons for signing up for 1Password, saving the item in 1Password, or accepting an invite to join the team from which the item was sent. Clicking those buttons will trigger a request to the item sharing server which is used to keep count of such clicks. Only overall and aggregate counts of the triggering events are saved server side.

7.5 Caveats

The item sharing service is intended to share copies of items with individuals who are not members of the 1Passsword family or team in which the original item resides. Within team or family accounts, sharing offers security properties which sharing via item sharing cannot.

Within an account, a relationship can be established between the personal key set of the members of accounts. But when sharing outside of an account, there's no preestablished relationship that can be used to identify the sender or the recipient. For this reason, item sharing senders and recipients need to take more care to verify independently that the other party is who they say they are.

When the recipient is given a share link through some channel, 1Passsword can't tell them whether the person sending the share link is the person who created the share link. Similarly, email authentication only proves the recipient had the ability to read an email message sent to that address at some time. Without the shared item remaining a single item shared within an account, there's far less ability to manage, control, or monitor use of that item as there would be within a team.

The identification of the item shared is entirely under the control of the sending client and can't be enforced by the 1Passsword server. A sender can evade the policy controls about which items they are allowed to send in much the same way a user can evade other client enforced permissions, such as having the ability to reveal a password. Thus a malicious user with the skills to modify their own client can share any item they're capable of decrypting by telling the 1Passsword server that some other permissible item was shared.

In practice, the channels over which the share link is transmitted lack the ideal security properties but they may well be sufficient for the needs at the time. Item sharing is safer and more convenient than the practical alternatives available to most users. In particular sharing with item sharing offers the ability to:

- Set an expiry time on the share.

- Limit the number of times it can be retrieved from the 1Passsword server

- Restrict retrieval to holders of a specific email address.

- Track what has been shared.

- Have policies stating what's allowed to be shared.

These abilities create a substantial security improvement against the practice of simply sharing unencrypted data over the same channels one would use for sending the share link.

We offer the item sharing service because it's enormously more secure than other ways people find to share 1Passsword items outside their family or team. After all, any 1Passsword user in a position to decrypt an item already has the ability to make a copy of its data and transmit that in any form of their choosing. Item sharing can't prevent insecure sharing, but it makes it easier for people and organizations to share copies of secrets in a more secure manner than they may otherwise.

Chapter 8

A deeper look at keys

It doesn't matter how strong an encryption algorithm is if the keys used to encrypt the data are available to an attacker or easily guessed. This section describes not only the details of the encryption algorithms and protocols used, it also covers how and where keys are derived, generated, and handled.

Here we provide details of how we achieve what is described in "How vault items are secured" and "How vaults are shared securely." It's one thing to assert (like we have) that our design means we never learn your secrets, but it's another to show how this is the case. As a consequence, this section will be substantially more technical than others.

8.1 Key creation

All keys are generated by the client using Cryptographically Secure Pseudo-Random Number Generator (CSPRNG). These keys are ultimately encrypted with keys derived from user secrets. Neither user secrets nor unencrypted keys are ever transmitted.

Table 8.1: *Random number generators used within 1Password by platform. The Browser platform refers to both the web client and to the 1Password X browser extension. "CLI" refers to the command line interface, op.*

Platform	Method	Library
iOS/macOS	SecRandomCopyBytes()	iOS/OS X Security
Windows	CryptGenRandom()	Cryptography API: NG
Browser	getRandomValues()	WebCrypto
Android	SecureRandom()	java.security
CLI	crypto/rand	Go standard crypto library

> **Dangerous bend**
>
> The public/private key pairs are generated using WebCrypto's crypto.generateKey as an RSA-OAEP padded RSA key, with a modulus length of 2048 bits and a public exponent of 65537.

> **Dangerous bend**
>
> The secret part of the Secret Key is generated by the client as we would generate a random password. Our password generation scheme takes care to ensure each possible character is chosen independently and with probability equal to any other. Secret Key characters are drawn from the character set {2-9, A-H, J-N, P-T, V-Z}.

> **Dangerous bend**
>
> An Elliptic Curve Digital Signature Algorithm (ECDSA) key is also created at this time. It's not used in the current version of 1Password, but its future use is anticipated. The key is generated on curve P-256.

8.2   Key derivation

For expository purposes, it's easiest to work backwards. The first section will discuss what a typical login looks like from an already enrolled user using an already enrolled client, as this involves the simplest instance of the protocol.

8.2.1   Deriving two keys from two secrets

Key Derivation

1: $p \leftarrow$ Account password from user
2: $(k_A, e, I, s) \leftarrow$ Secret Key, email address, ID, salt from local storage
3: $p \leftarrow \textbf{trim}(p)$
4: $p \leftarrow \textbf{normalize}(p)$
5: $s \leftarrow \textbf{HKDF}(s, \texttt{version}, e, 32)$
6: $k_m \leftarrow \textbf{PBKDF2}(p, s, 650000)$
7: $k_A \leftarrow \textbf{HKDF}(k_A, \texttt{version}, I, \|k_m\|)$
8: $k_m \leftarrow k_m \oplus k_A$
9: $k_m \leftarrow \textbf{JWKify}(k_m)$

Figure 8.1: *The AUK is derived from the account password, p; Secret Key, kA, and several non- secrets including the account ID, I, and a salt, s.*

As discussed in "Account password and Secret Key," 1Password uses two-secret key derivation (2SKD) so data we store cannot be used in brute-force cracking attempts against a user's account password. The two secrets held by the user are their account password and Secret Key.

From those two user secrets the client needs to derive two independent keys. One is the key needed to decrypt their data, the other is the key needed for authentication. We'll call the key needed to decrypt the data encryption keys (and, in particular, the user's private key) the Account Unlock Key (AUK), and the key that's used as the secret for authentication SRP-x.

The processes for deriving each of these are similar, but involve different salt in the key derivation function. A user will have a salt used for deriving the Account Unlock Key (AUK) and a different salt used for deriving SRP-x.

In both cases, the secret inputs to the key derivation process are the user's account password and Secret Key. Non-secret inputs include salt, algorithm information, and the user's email address.

8.2.2   Preprocessing the account password

Before the user's account password is handed off to the slow hashing mechanism, it undergoes a number of preparatory steps. The details and rationale for those are described here.

account password are first stripped of any leading or trailing whitespace (Step 3, Figure 8.1). Whitespace is allowed within the account password, but because leading or trailing whitespace

may not be visible to the user, we want to avoid creating an account password with spaces they're unaware of. Then it's normalized (Step 4) to a UTF-8 byte string using Unicode Normalization Form Compatibility Decomposition (NFKD). By normalizing these strings before any further processing, we allow for different byte sequences that may encode the same Unicode character to be treated identically. This is discussed in greater depth in Discussion 9.1.

> **Discussion 9.1: Non-ASCII passwords**
>
> People naturally want to use passwords that involve characters other than the 7-bit US-ASCII printable characters. Yet doing so poses difficulties that simply supporting Unicode doesn't answer. Unicode normalization goes a long way toward addressing these difficulties. The need for Unicode normalization can be exemplified by considering how the glyph "Å" may be encoded when it's typed on some devices. It can be encoded in (at least) three different ways: It might be the byte sequence 0x212B, or 0x00C5, or 0x0041030A. Exactly how it's encoded and passed to 1Password (or any software) depends on the often arbitrary details of the user's keyboard, operating system, and settings. 1Password itself has no control over what particular sequence of bytes it will receive, but the user who uses "Å" in their password needs it to work reliably.
>
> Normalization ensures that whichever particular UTF encoding of a string is passed to 1Password by the user's operating system will be treated as identical. In the case of "Å", the normalization we have chosen (NFKD), will convert any of those three representations to 0x0041030A.

> **Dangerous bend**
>
> Normalization does not correct for homoglyphs. For example, the letter "a" (the second letter in "password") in the Latin alphabet will never be treated the same as the visually similar letter "а" (the second letter in "пароль") in the Cyrillic alphabet. Thus, despite our use of normalization, users still have to exercise care in the construction of account passwords that go beyond the unambiguous 7-bit US-ASCII character set.

8.2.3 Preparing the salt

Next, the 16-byte salt is stretched using hash-based key derivation function (HKDF) and salted with the lowercase version of the email address (Step 5).[11] The reason for binding the email address tightly with the cryptographic keys is discussed in "Restoring a user's access to a vault."

8.2.4 Slow hashing

The normalized account password is then processed with the slow hash PBKDF2-HMAC-SHA256 along with a salt.

---

[11] HKDF places no security requirements on its salt, which may even be a constant or zero.

> **Dangerous bend**
>
> The choice of PBKDF2-HMAC-SHA256 as our slow hash is largely a function of there being (reasonably) efficient implementations available for all our clients. While we could have used a more modern password hashing scheme, any advantage of doing so would have been lost by how slowly it would run within JavaScript in most web browsers.
>
> Because key derivation is performed by the client (so the server never needs to see the password) we are constrained in our choices by our least efficient client. The Makwa password hashing scheme [a], however, is a possible road forward because it allows some of the computation to be passed to a server
>
> ---
> [a]@pornin:MAKWA

In the current version, there are 650,000 iterations of PBKDF2.[12] Extrapolating from a cracking challenge we ran,[13] we estimate it costs an optimized attacker working at scale between 30 and ~40 US dollars to make 2^{32} guesses against PBKDF2-SHA256 with 650,000 iterations.

8.2.5   Combining with the Secret Key

```
{    "alg":"A256GCM",
     "ext":false,
     "k":"KCVXrFs8oJBheco-JxbHkPL9bxyN5\
         WZ68hyfVgrBuJg",
     "key_ops":["encrypt","decrypt"],
     "kty":"oct",
     "kid":"mp"}
```

Figure 8.2: *The AUK is represented as a JSON Web Key (JWK) object and given the distinguished key ID of mp.*

The Secret Key, treated as a sequence of bytes, is used to generate an intermediate key of the same length as that derived from the account password. This is done using hash-based key derivation function (HKDF), using the raw Secret Key as its entropy source, the account ID as its salt, and the format version as additional data.

The resulting bytes from the use of HKDF are XORed with the result of the PBKDF2 operations. This is then set with the structure of a JSON Web Key (JWK) object as illustrated in Figure 8.2.

8.2.6   Deriving the authentication key

The process of deriving the client-side authentication secret used for authenticating with the 1Password server is nearly identical to the procedure described above for deriving the Account Unlock Key (AUK). The only difference is an entirely independent salt is used for the PBKDF2 rounds. This ensures the derived keys are independent of each other.

The 32-byte resulting key is converted into a BigNum for use with Secure Remote Password (SRP). We use the JSBN library in the browser, and the tools from OpenSSL for all other platforms.

The astute reader may have noticed the defender needs to perform 1,300,000 PBKDF2 rounds while an attacker (who has managed to obtain the Secret Key) only needs to perform 650,000

---

[12]Accounts created prior to January 27, 2023 and have not changed their account password or Secret Key since this date, will use a lower iteration count. The iteration count can be updated to the current standard value by changing either the account password or Secret Key.

[13]Goldberg (2021)

PBKDF2 rounds per guess, thus giving the attacker a 1-bit advantage over the defender in such an attack.

The sequence described above, however, in which the defender needs to derive both keys, rarely happens. In most instances, the SRP-x will be encrypted with the Account Unlock Key (AUK) (or by some other key that's encrypted with the AUK) and stored locally. Thus the defender needs to derive the AUK only. The client needs to go through both derivations only at original sign-up or when enrolling a new client.

8.3 Initial sign-up

To focus on the initial creation of keys and establishment of authentication mechanisms, this section assumes the enrolling user has been invited to join a team by someone authorized to invite them.

When the invitation is created, the server generates an account ID and knows which team someone has been invited to join and the type of account that's being created. The server is given the new user's email address and possibly the new user's real name. An invitation Universally Unique Identifier (UUID) is created to uniquely identify the invitation, and known to the team administrator. An invitation token is created by the server and not made available to the administrator. Other information about the status of the invitation is stored on the server.

The user is given (typically by email) the invitation Universally Unique Identifier (UUID) along with the invitation token, and uses them to request invitation details from the server. If the UUID is for a valid and active invitation and the provided token matches the invitation's token, the server will send the invitation details, which include the account name, invited email address, and (if supplied by the inviter) real name of the user. If the server doesn't find a valid and active invitation for that UUID, it returns an error.

The client will gather and compute a great deal of information, some of which is sent to the server.

Table 8.2: *Symbols used to indicate status of different data client creates during signup.*

Symbol	Meaning
ξ	Generated randomly
🔑	Key-like thing
🔒	Encrypted
↑	Uploaded

1. Generate Secret Key ξ 🔑
2. Compute AUK
    1. Generate encryption key salt ξ ↑
    2. Derive Account Unlock Key (AUK) from encryption salt, account password, and Secret Key as described in "Key derivation." 🔑
3. Create encrypted account key set
    1. Generate private key ξ 🔑
    2. Compute public key (from private key) 🔑 ↑
    3. Encrypt private part with AUK 🔒 ↑

    4. Generate key set Universally Unique Identifier (UUID) ξ ↑

    5. Include key set format ↑

4. User information ↑

    1. Given name ↑

    2. Family name ↑

    3. Avatar image ↑

    4. Email address ↑

5. Device information ↑

    1. Generate device UUID ξ ↑

    2. Operating system (if available) ↑

    3. User agent (if applicable) ↑

    4. Hostname (if available) ↑

6. Construct Secure Remote Password (SRP) verifier

    1. Generate authentication salt ξ ↑

    2. Derive SRP-x from account password, Secret Key, an authentication salt 🔑

    3. Computer SRP verifier from SRP-x 🔑 ↑

7. Send to the server everything marked ↑

8.3.1   Protecting email invitations

Invitations are sent by email, and suffer the security limitations of email. Administrators are strongly encouraged to verify independently (by means other than email) the intended recipients have enrolled.

8.4   Enrolling a new client

When enrolling a new device, the user will provide the client with the add-device link (possibly in the form of a QR code) and their account password. The add-device link is generated at the user's request from an already enrolled client and includes the domain name for the team, the user's email address, and their Secret Key.

The link uses the custom schema `onepassword:` with a path of `//team-account/add` and a query string with fields `email`, `server`, and `key`. An example is shown in Figure 8.3.

```
onepassword://team-account/add?email=patty@dogs.example
&server=https://example.1password.com&key=A3-8MMQJN-MZ64CY-2SDB4-RPX3T-V52Q3-N2C84
```

Figure 8.3: *An add link contains the email address, team domain, and Secret Key.*

This new client doesn't have its salt nor its key derivation parameters so requests them from the server. It's able to generate its device information and create a device Universally Unique Identifier (UUID).

```
{ "accountKeyFormat" : "A3",
  "accountKeyUuid" : "GWM4R8",
  "sessionID" : "TRYYDRPO2FDWRITHY7BETQZPN4",
  "status" : "ok",
  "userAuth" : {
      "alg" : "PBES2g-HS256",
      "iterations" : 650000,
      "method" : "SRPg-4096",
      "salt" : "WSwigQtQpxqYAri592W1lg"
  } }
```

Figure 8.4: *Example response from server to auth request. The Secret Key is often referred to as "Account Key" internally.*

The client initiates an auth request to the server, sending the email address and device Universally Unique Identifier (UUID). A typical server response looks similar to what's shown in Figure 8.4.

After the client has the salt used for deriving its authentication secret, it can compute its SRP-x from that salt, the account password, and the Secret Key. During authentication, neither the client nor server reveals any secrets to the other, and after authentication is complete, our own transport layer encryption is invoked on top of what is provided by Transport Layer Security (TLS). In the discussion here, however, we'll ignore those two layers of transport encryption and present the data as seen by the client and server after both transport encryption layers have been handled.

After successful authentication, the client requests its encrypted personal key set from the server. If the client has successfully authenticated, the server allows it to fetch the key sets associated with the account. The personal key set has the overall structure shown in Figure 8.5.

```
{"encPriKey": {
 "..." },
 "encSymKey": { "..." },
 "encryptedBy": "mp",
 "pubKey": { "..." },
 "uuid" : "c4pxet7a..."}
```

Figure 8.5: *Overview of personal key set. The value of 'encryptedBy' here indicates the encrypted symmetric key is encrypted with the Account Unlock Key.*

This contains an encrypted private key, the associated public key, and an encrypted symmetric key that's used to encrypt the private key. The encrypted symmetric key is encrypted with the Account Unlock Key (AUK), using the parameters and salt that are included with the encrypted symmetric key as shown in Figure 8.6

```
{ "encPriKey" : { "..." },
  "encSymKey": {
       "kid": "mp",
       "enc": "A256GCM", "cty": "b5+jwk+json",
       "iv": "X3uQ83tlrdNIT_MG",
       "data": "gd9fzh8lqq5BYdGZpypXvMzIfkS ...",
       "alg": "PBES2g-HS256", "p2c": 650000,
       "p2s": "5UMfnZ23QaNVpyeKEusdwg" },
  "encryptedBy" : "mp",
  "pubKey" { "..." }, "uuid" : "c4pxet7a..." }
```

Figure 8.6: *The encrypted symmetric key is encrypted with the AUK, which in turn is derived using the salt in the p2s field, and using the methods indicated in the fields alg and p2c. The encrypted symmetric key itself is encrypted using AES256-GCM.*

The details of the public and private keys are illustrated in Figure 8.7.

```
{ "encPriKey" :    {
       "kid":   "c4pxet7agzqqhqg9yvxc2hkg8g",
       "enc":   "A256GCM", "cty": "b5+jwk+json",
       "iv":    "dBGJAy3uD4hJkf0K",
       "data": "YNz19jMUffFP_glxM5Z ..."},
  "encSymKey": { "..." },
  "encryptedBy" : "mp",
  "pubKey" {
       "alg": "RSA-OAEP-256", "e": "AQAB",
       "key_ops": ["encrypt"], "kty": "RSA",
       "n": "nXk65CscbXVuSq8I43RmGWr9eI391z ...",
       "kid": "c4pxet7adzqqvqg9ybxc2hkg8g" },
  "uuid" : "c4pxet7agzqqhqg9yvxc2hkg8g"  }
```

Figure 8.7: *The public/private parts are specified using JWK.*

8.5   Normal unlock and sign-in

When you unlock and and sign in to 1Password from a client that has previously signed in, the client may[14] have everything it needs locally to compute its Account Unlock Key (AUK) and to compute or decrypt SRP-x. The client may already have the salt, encryption parameters, and its encrypted personal key set.

After the user enters a correct account password and the client reads the Secret Key, it computes the Account Unlock Key (AUK), decrypts the user's private key, then decrypts any locally cached data. Depending on the completeness of the cached data, the client may be able to function offline.

---

[14]The use of the word "may" here reflects the fact that different 1Password clients take different approaches to what they store locally and what they recompute. The current version of the web client, for example, caches much less data locally than the mobile clients do.

Chapter 9

Unlock with a passkey or single sign-on

As an alternative to the sign-in method described in chapter "A deeper look at keys," it's also possible to sign in to 1Password with a passkey or single sign-on (SSO) provider.

Anyone can create an account that uses a passkey for authentication. When you set up an account this way, you provide your account's passkey to unlock 1Password instead of an account password and Secret Key.

Companies that use 1Password can configure unlock with single sign-on (SSO) for groups in their organization. When a user signs in with SSO, they sign in with the username, password, and other authentication factors required by their SSO provider instead of using their account password and Secret Key to authenticate to 1Password. 1Password accepts proof of authorization from the SSO provider as authentication.

9.1   Unlocking without an account password

We designed passkey and single sign-on (SSO) unlock to work similarly to signing in with an account password and Secret Key in that both methods set up a process that uses Secure Remote Password (SRP) in a similar way. On devices where a user signs in with a passkey or SSO, 1Password clients store a device key. Each device key is uniquely and randomly generated, and never leaves the device on which it was created. To enroll a new device on a passkey or SSO-enabled account, the user must authentication first then authorize the new device using a previously enrolled device.

> **Device key**
>
> A cryptographic key stored on a 1Password client that's using single sign-on (SSO). It's used to decrypt the credential bundle it receives from the server upon successful sign in.

With passkey and single sign-on (SSO) unlock, your first device randomly generates an SRP-x and Account Unlock Key (AUK). They're stored on our servers, encrypted by the device key that's only stored on the device that created it. This combination of the SRP-x and AUK is called a credential bundle.

> **Credential bundle**
>
> Consists of a randomly generated SRP-x and AUK, it's used to sign in to 1Password with single sign-on (SSO). It's encrypted by the device key and stored on 1Password servers.
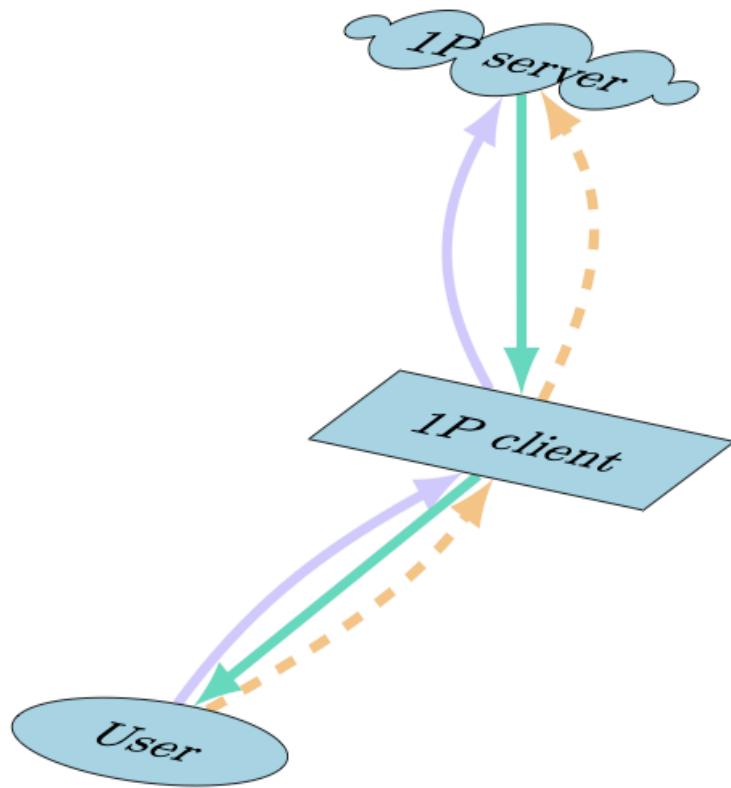
Figure 9.1: *Passkey sign in. The solid purple arrows illustrate the authorization of a device when a user performs a passkey sign-in, green arrows illustrate the return of the credential Bundle to the user, and dashed golden arrows illustrate the user's authentication with SRP to use 1Password.*
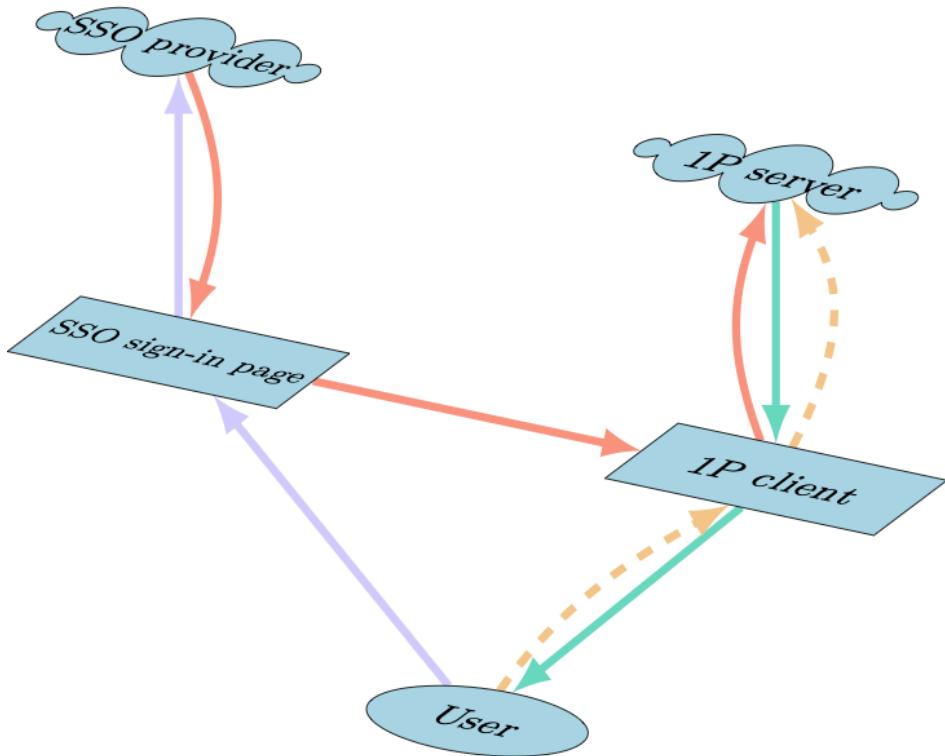
Figure 9.2: *Single sign-on sign in. The solid purple arrows indicate a user signing in to their SSO provider. The solid red arrow shows the authorization the SSO provider sends to the 1Password server. The green arrows show the credential bundle being returned to the user. The dashed golden arrows show the user authenticating with SRP to use 1Password. This diagram is based on the OpenID Connect SSO authorization flow. For some SSO providers, the destinations of certain arrows may be slightly different.*

9.1.1 Authorization and the credential bundle

Authorization to obtain the credential bundle happens as follows:

- **Passkey unlock** The server authentication your passkey and authorizes you.

- **SSO unlock** During sign-in, the single sign-on (SSO) provider tells 1Password servers you've successfully authorized. Typically, the SSO provider returns an authorization token to your device, which forwards it to the 1Password server.

In return for a valid proof of authorization, our servers return a credential bundle encrypted with the device key. After the 1Password client decrypts the SRP-x and Account Unlock Key (AUK) with the device key, it authenticates as described in "A deeper look at keys." After successful sign-in with a passkey or an SSO provider, 1Password behaves identically to when an account password and Secret Key are used.

9.2 Linked apps and browsers

After you've successfully enrolled with a passkey or single sign-on (SSO), the app or browser you use is linked. The device you use stores a device key and sets up a unique credential bundle. The first client used to signed in to 1Password – either for the first time or after a user has been restored – is a linked app (or browser) by default.

> **Linked app or browser**
>
> A client trusted to use SSO, by having set up a device key and created a corresponding credential bundle.

The first app or browser you use to sign in creates a new set of randomly generated values that form the credential bundle. Any additional apps or browsers you enroll need approval. They're approved by successfully authenticating to the single sign-on (SSO) provider, consenting to the sign-in with an existing linked app, and providing a code that's randomly generated by the linked device.

When you approve a sign-in within your linked app or browser, it sends a copy of the credential bundle to the new device via an end-to-end (E2E) encrypted channel. The new app protects the credential bundle with its own unique device key. The device key is critical for the overall security of single sign-on (SSO). Appendix A has more information about device key security and storage.

9.3   Linking other devices

When you set up a new app or browser, the credential bundle the device uses is obtained from a previously linked app or browser. For your existing device to send the credential bundle to your new device, a trusted channel is set up between the two devices. For reliability, that channel is facilitated by 1Password servers and set up in such a way that 1Password can't see what the two devices are sending each other.

The trusted channel between two devices uses the CPace cryptographic protocol. With CPace, two devices with knowledge of a six-character code can authentication to one another and agree on a shared encryption key. That encryption key is used to encrypt the credential bundle when it's sent from a linked device to a new device which makes the contents impossible to decrypt for anyone observing the encrypted messages. In the event a malicious server attempts to interfere in the key agreement process, 1Password clients detect the presence and abandon participation.

> **CPace**
>
> A modern PAKE using a shared secret, defined by Abdalla, Haase, and Hesse (CPace, a balanced composable PAKE.)[a]
>
> ---
> [a]@DraftCpaceRFCv07

With these building blocks, the process shown in Figure 9.3 and annotated below describes how a credential bundle safely travels between two devices.
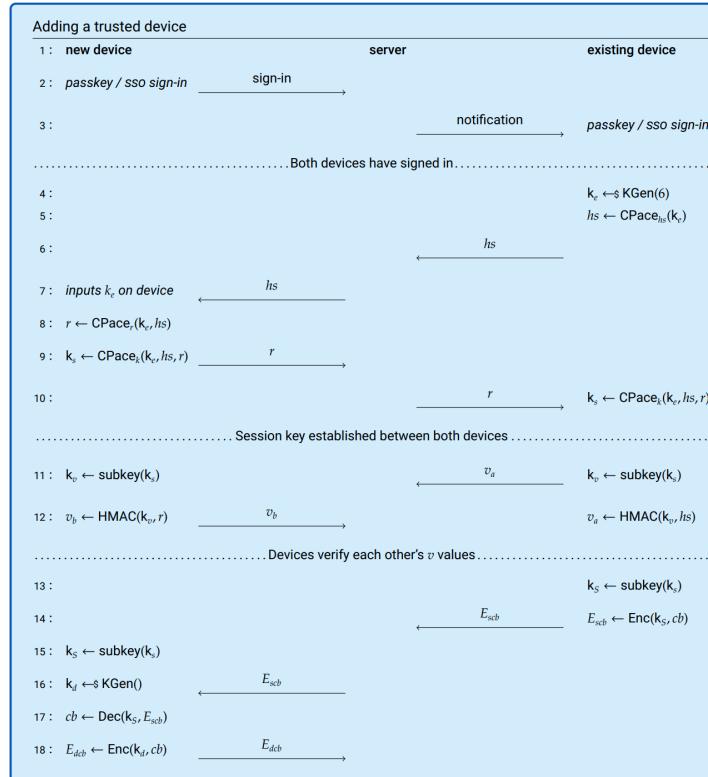
Figure 9.3: *An overview of the protocol by which a linked app or browser is added, showing the communication between the linked client, new device, and 1Password server. Any SSO providers that perform initial sign-in are not depicted.*

Line 1: The parties involved are a new device, the server, and a linked app or browser. Your single sign-on (SSO) provider, if applicable, also plays a small role initially but they're omitted from the figure for simplicity.

Line 2: You sign in to 1Password with a passkey or SSO.

Line 3: The 1Password server sends a notification to all existing linked apps and browsers. They'll notify you that a new device wants to be set up and you need to approve the connection on the existing device. If you choose to continue, you'll have to sign in on the existing device unless you already have an active session.

Lines 4-7: Your existing device initiates a trusted channel with the new device using CPace. The existing device then generates a 6-character setup code, uses it to create a CPace handshake hs, and sends the handshake to the 1Password server.

Lines 7-10: Your new device fetches the CPace handshake and asks you to enter your setup code. After you enter the setup code, your device computes a CPace *reply* r from information in both the CPace handshake and the setup code, then sends it to the 1Password server.

Both devices use the shared values to compute a shared session key k_s.

Lines 11-13: Before the keys are used, it's important to verify the keys have been exchanged correctly. After all, you may have accidentally entered the wrong setup code or there may have been something nefarious that tried to influence the messages sent between your devices.

To verify the keys, both devices compute an HMAC digest of the message they received from the other device using the key they both derived. They send these verification values to one another and verify whether the value computed by the other matches their own. If the values don't match on either device they break off the setup process and start over again.

Lines 13-18: Your linked app or browser encrypts the credential bundle with (a derivative of) the session key established previously and sends them to your new device via the 1Password server. Your new device derives the decryption key the same way and decrypts the credential bundle. Next, the device generates a random device key, stores it, then encrypts the credential bundle with the device key. Your device stores the newly encrypted credential bundle on the 1Password server and completes the process to become a linked app or browser.

9.4   Quick on-device access with biometrics

The process described in Figure 9.2 requires that your device be online when unlocked. It's possible for certain devices to get access to vault contents while offline if the user's business account is configured to allow it. Offline access to vault contents is provided when a user successfully performs a biometric authentication. This is supported on Windows, Linux, macOS, iOS and Android using their respective platform's biometric authentication.

When you unlock with biometrics, the credential bundle is used to decrypt vault contents locally so it can be accessed offline. Clients also keep track of a reauthentication token. This token is used to perform reauthentication with the 1Password server within a limited timeframe, without the client performing passkey unlock or reaching out to the SSO server. When an account administrator turns on biometric unlock, they temporarily delegate the responsibility of authenticating you to your device instead of your identity provider.

A reauthentication token is requested when you use biometrics to unlock your passkey or single sign-on (SSO)-enabled 1Password account. It's guarded by the protections described in "Transport security" when it's transferred from the 1Password server to your device.

On macOS, iOS and Android devices, quick biometric unlock is protected by the respective platform's built-in secure elements. On Windows and Linux, the reauthentication token is stored in protected operating system memory while the 1Password app is running either when locked or unlocked. On the platforms that store the reauthentication token in memory, the token is lost when the app closes or restarts, so you need to sign in to 1Password again.

Chapter 10

Revoking access

When Alice tells Bob a secret and later regrets doing so, she can't make Bob forget the secret without resorting to brain surgery. We feel brain surgery is beyond the scope of 1Password,[15] and therefore users should be aware that once a secret has been shared the recipient cannot be forced to forget that secret.

> ### Story 7: A week in the life of revocation
>
> We're always happy for our colleagues when they move on to new adventures.
>
> Tom and Gerry have been working on Widgets For Cows, Barnyard Gadgets' new Internet of Things products, and it's time for Tom to move on. Tom will get access to a new team and new shared vault.
>
> Ricky, the team owner, adds Tom to the new vault. Adding a new member to a shared vault is very simple. A copy of the vault key will be encrypted with Tom's public key so only Tom can decrypt it, and Tom will be sent a notification about the new shared vault. But what about his old access and Gerry's new product plans for Widgets for Cows?
>
> Ricky will remove Tom from the Widgets for Cows vault. Ricky can't make Tom forget information that he's already had and perhaps made a copy of, but Tom can be denied access to anything new added to the vault.
>
> After Tom has been removed from the vault, Gerry creates a new Document called *More Cow Bell* for the vault. *More Cow Bell* will be encrypted with a key that's encrypted by the vault key, but Tom should never get a copy of the encrypted Document item.
>
> The next time Tom connects to the server, he will no longer be sent data from that vault. This server policy mechanism prevents Tom from receiving any new data from that vault. Furthermore, Tom's client will be told to remove any copies of the vault key and the encrypted data it has stored for that vault. This client policy at least get a well behaved client to forget data and keys it should no longer have. Either of those policies is sufficient to prevent Tom from learning just how much cow bell Gerry thinks is enough.

---

[15]We've made no formal decision on whether rocket science is also beyond its scope.

Chapter 11

Access control enforcement

Users (and attackers) of 1Password are limited in what they can do with the data. Enabling the right people to see, create, or manipulate data while preventing the wrong people from doing so is the point of 1Password. The sorts of powers that an individual has are often discussed in terms of Access Control Lists (ACLs). For want of a better term, we'll use that language here; however, it should be noted that the mechanisms by which these controls are enforced aren't generally the same as the ones for more traditional ACLs. Indeed, different controls may be enforced by different mechanisms, even if presented to the user in the same way.

Broadly speaking, there are three kinds of control mechanisms. These are cryptographic enforcement of policy, server enforcement of policy, and client enforcement of policy.

11.1 Cryptographically enforced controls

If someone hasn't been given access to a vault, it's impossible in all practical terms for them to decrypt its data. So at the simplest level, if a user hasn't been added to a vault, the mathematics of cryptography ensure they won't be able to decrypt it.

Because the server never has access to decrypted vault keys, it can't give out those keys to anyone. Therefore the server simply doesn't have the power to grant someone access to a vault. Such requirements are cryptographically enforced.

Among the mechanisms cryptographically enforced:

- Unlocking a vault.

- Only those with access to a vault can share it.

- User email address can be changed only by the user.

- Server doesn't learn user's Secret Key or account password.

11.2 Server-enforced controls

Cryptography doesn't prevent a user (or their client) with access to the vault key from adding, deleting, or modifying items in that vault when the information resides locally on their device. The same key they use to decrypt the data could be used to encrypt modified data.

But 1Password offers the ability to grant individuals read permission to a vault while denying them write permission. The server will reject any attempt from a read-only user of a vault to upload data to that vault. This, and other features, are enforced by server policy. An example of one of these in action is presented in Story 8.

> **Story 8: A day in the life of read-only data**
>
> Patty (a clever and sneaky dog) has been granted access to a vault called "Locations" that contains the locations of the water dish and the dog door. So has another member of the team, Morgan le Chien.
>
> Patty thinks she will have the place to herself if Morgan can't manage to settle in. So she'd like to give Morgan misleading information. Although Patty has been granted only read access to the Locations vault, she is a remarkably clever dog and extracts the vault key from her own data. The same vault key that decrypts items is also used to encrypt items. She modifies the location of the water bowl (listing the driest part of the house) and encrypts her modified data with the vault key. Then she tries to send this modified data to the server so Morgan will get that information instead.
>
> But she finds that **server policy** prevents her from uploading modified data. Although cryptographically she had the ability to modify the data, she could only do so on her system. Her evil plan was foiled by server policy.
>
> Of course, her plan would have failed anyway. Morgan is happy to drink from anything resembling a water receptacle, and can manage remarkably well even if she doesn't know the location of the water bowl.

11.3 Client-enforced controls

Client-enforced controls are limitations enforced within either the web browser or a native client, such as an iOS application. Because the web browser or native client is running on a user's system and outside our control, these policies may be circumvented by a malicious client or determined user. This doesn't reduce their usefulness to ordinary users and may help prevent unintended disclosures or accidental actions.

See Story 9 for an illustration of what client-enforced policies can and can't do.

11.3.1 Controls enforced by client policy

Each of the client policies requires a server or cryptographically enforced policy be granted in order to be allowed. For example, the Import permission may be circumvented by a client, but the user will be unable to save the newly imported item to the server because the Write permission is enforced by the server, not the client.

- **Importing items into a vault.** A user may still create multiple items manually provided they have permission to create new items in the vault. This permission may be used to restrict how many items a user may easily create or prevent accidentally importing items.

- **Exporting items from a vault.** A user may still obtain the item data by other means and create files that aren't controlled by 1Password. This permission may be used to prevent accidentally disclosing the contents of an entire vault.

- **Revealing a password for an item.** A user may still obtain the password by examining a web page using the developers' tools for their web browser. This permission may be used to prevent accidental disclosure and may help reduce the risk of shoulder surfing and other social engineering attacks.

- **Printing one or more items.** A user may still obtain the item data by other means; create files that are not controlled by 1Password and print out those files. This permission may be used to prevent accidentally disclosing the contents of an entire vault.

11.4 Multiple layers of enforcement

Something enforced by cryptography may also be enforced by the server, and something enforced by the server may also be enforced by the client. For example, the server won't provide the vault data to non-members of a vault, even though non-members wouldn't be able to decrypt the data even if it were provided. Likewise, a 1Password client will generally not ask for data the server would refuse to supply. Throughout this document, we'll typically mention the deepest layer of enforcement only.

Story 9: A day in the life of a concealed password

The administrators have come to be wary of how the dog Patty (see Story 8 for background) treats data. They want Patty to have access to the password for the dog door (they want her to be able to leave and enter as she pleases), but they don't want Patty to give the password to any of her friends should her paws accidentally press the Reveal button.

So the administrators limit Patty's ability to reveal the password. She can fill it into the website that controls the dog door (she lives in a somewhat unusual household), but she can't accidentally press 1Password's Reveal button while her friends are watching. This is protected by client policy.

But Patty is a clever dog. After she uses 1Password to fill on the website, she uses her browser's debugging tools to inspect what 1Password has inserted. She gets the password, and tells all her friends so they can come and visit.

The house is suddenly full of Patty's friends running wild, and the administrators have learned an important lesson: Client policy controls are easily evaded.

Chapter 12

Restoring a user's access to a vault

If Albert forgets or loses their Secret Key or account password, it's impossible to decrypt the contents of their vaults unless those vaults have been shared with someone else who hasn't forgotten or lost the Secret Key or account password. Our use of two-secret key derivation (2SKD) increases the risk to data availability because, in addition to the possibility of a user forgetting their account password, there's also the possibility the Secret Key gets lost. Data loss can be catastrophic to a team, so some recovery mechanism is necessary.

Our security design also requires that we at 1Password never have the ability to decrypt your data, so we don't have the ability to restore anyone else's ability to decrypt their data if they have forgotten their account password or lost their Secret Key. Our solution is to place the power to recover access to vaults where it belongs: within the team.

12.1   Overview of groups

To understand how the Recovery Group works, it's first necessary to understand how a group works. A group will have a key set that's similar in nature to an individual's key set. It's an encrypted public/private key pair. A vault is held by a group if the vault key is encrypted with the group's public key.

> **Dangerous bend**
>
> An individual (or another group) is a member of the group if the group's private key can be decrypted by that individual. To put it simply[a] A is a member of group G if and only if G's private key is encrypted with A's public key. A can decrypt anything encrypted with her public key because she can decrypt her private key. Thus, A will be able to decrypt the private key of G. With G's private key, she can decrypt the vault keys that are encrypted with G's public key. But if A hasn't been granted access to a vault, she'll be prevented by server policy from obtaining the vault data even though she has the key to that vault. Simple.
>
> ---
> [a]For some values of the word "simply."

12.2   Recovery groups

One of the most powerful capabilities a team administrator has is the power to assign members to the team's Recovery Group. In most configurations the assignment is automatic and Owners, Organizers, and Administrators will automatically be made members of the group. In 1Password Families there's no ability to separate the roles of Owner, Administrator, and Recovery Group member; they're all wrapped up as "Organizer." With 1Password Teams, Administrators are given more control, but not all the underlying flexibility may be exposed to the user.[16] This document describes recovery in terms of the Recovery Group even when the group is not exposed to the Team administrator in those terms.

---

[16]We discovered during our beta testing that it was difficult to make the distinction between Owners, Administrators, vault Managers, and Recovery Group members clear enough for those distinctions to be sufficiently useful.

12.3   Implicit sharing

When a vault is created, a copy of the vault key is encrypted with the public key of the recovery group. The members of the Recovery Group can decrypt the private key of the recovery group. Thus from an exclusively cryptographic point of view, the members of the Recovery Group have access to all the vaults.[17]

Recovery Group member never have the ability to learn anyone's account password, Secret Key, Account Unlock Key (AUK), or SRP-x. Recovery is recovery of the vault keys — it's not recovery of the account password or Secret Key.

12.4   Protecting vaults from recovery group members

Although there's a chain of keys and data that would allow any member of the Recovery Group to decrypt the contents of any vault, there are mechanisms that prevent it.

- A member of the Recovery Group won't be granted access to the encrypted data in a vault they otherwise wouldn't have access to, even if they can obtain the vault key.

- A member of a Recovery Group will only be sent the encrypted vault keys after the user requesting recovery has re-created their account.

Thus the server prevents a member of the Recovery Group from obtaining the vault keys without action on the part of the person seeking recovery. The capacity to decrypt the vault keys offers the malicious member of a recovery group little benefit if those encrypted keys are never provided. Furthermore, even if a malicious member of the recovery group can trick the server into delivering the encrypted vault keys when it shouldn't, the attacker still needs to obtain the vault data encrypted with that key.

12.5   Recovery risks

Recovery mechanisms are inherently weak points in maintaining the secrecy of data. Although we have worked to design ours to defend against various attacks, there are special precautions that should be taken when managing a Recovery Group or authorizing recovery.

- Members of a recovery group should be adept at keeping the devices they use secure and free of malware.

- Members of the recovery group should be aware of social engineering trickery.

- Recovery requests should be verified independently of email. (Face to face or a phone call should be used.)

- Recovery emails should be sent only if you have confidence in the security of the email system.

- If there are no members of a recovery group, the capacity to recover data is lost to the team.

---

[17]1Password Teams also have a permission called Manage All Groups that has equivalent cryptographic access, which is only given to the Administrators and Owners groups by default.
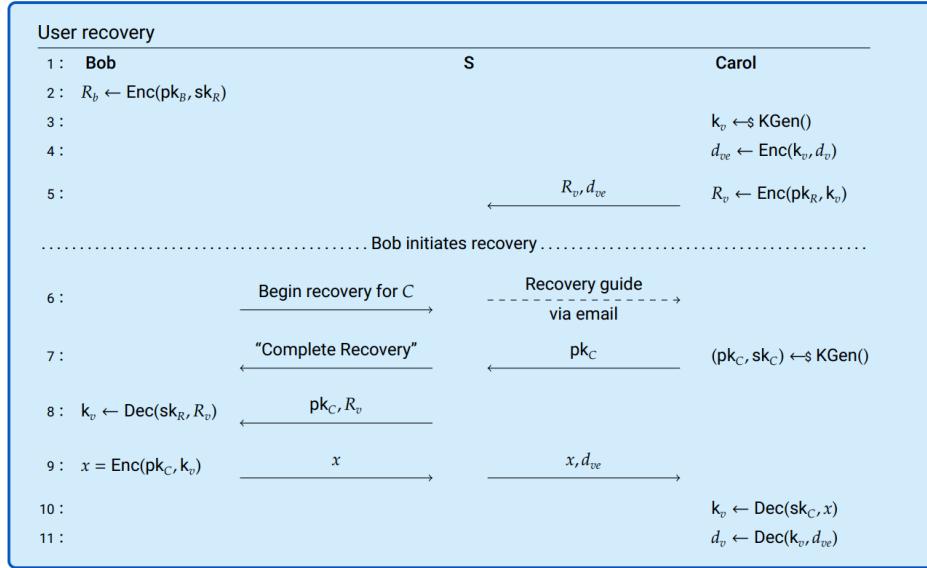
Figure 12.1: *An overview of what keys are available to whom and when to support data recovery.*

Figure 12.1 provides an overview of what data and keys are held by whom. Some hopefully irrelevant details have been omitted to keep the diagram manageable. For example, the transport encryption layers for the messages are entirely skipped (see "Transport security") and where we speak of encrypting or decrypting private keys, it's actually encrypting or decrypting the keys that the private parts of keys are encrypted with (see 5.1). The illustration acts as if Carol would only ever have a single vault, though of course she may create a number of different vaults.

Line 1: Our participants are Bob, a member of the Recovery Group; Carol, a member of the same team but not a member of the recovery group; and S, the 1Password Server.

Line 2: Bob starts with his one personal key set, $(\mathsf{pk}_B, \mathsf{sk}_B)$, and with the private key, sk_R, of the recovery group encrypted with Bob's public key.

Lines 3-4: Carol creates a new vault which will be encrypted using vault key k_v, which her client generates. Encrypting the items in a vault is properly described in Figure 5.1. Here we just abbreviate it as "$\mathsf{Enc}(\mathsf{k}, d_v)$".

Line 5: When Carol creates a vault a copy of its vault key, k_v, is encrypted using the recovery group's public key, pk_R, and sent to the server. The encrypted vault data, d_{ve}, is also sent to the server for syncing and storage.

Line 6: When Bob initiates recovery (presumably after receiving a request from Carol outside of this system, as Carol can no longer sign in), Bob informs the server of his intent, and the server sends instructions to Carol by email.

Not shown in this diagram is the server putting Carol's account into a specific suspended state. If Carol successfully signs in, the recovery is automatically cancelled.

Line 7: When Carol's account is in a pending recovery state, she's directed through a procedure very similar to initial signup. The key difference being that she maintains the same name, email address, and permissions instead of being treated as a new user by the system.

During this process, Carol generates a new personal key set, $(\mathsf{pk}_C, \mathsf{sk}_C)$, and shares her new public key, pk_C with the server.

The server will inform Bob that he needs to "complete the recovery" of Carol's account.[18]

---

[18]This would be a good time for Bob to confirm with Carol through some method other than email that it's indeed Carol

Not shown is Carol's client generating a new Secret Key during this recovery signup. Carol will choose a new account password, which may be identical to her previous one. And from her new Secret Key and potentially new account password, her client will generate a new Account Unlock Key (AUK) with which it will encrypt her new personal key set.

Line 8: After Carol has created her new key set and Bob confirms he wishes to complete recovery, the server will send Carol's new public key, pk_C, along with the copy of the vault key that's encrypted with the recovery group's key, R_v. Recall that R_v was sent to the server when Carol first created the vault.

Lines 8-9: Bob can decrypt R_v and re-encrypt it as x with Carol's new public key, pk_C and send that to the server.

The server can then pass x back to Carol, along with the encrypted data in the vault, d_{ve}.

There are several things to note about the process illustrated in Figure 12.1. Most importantly, at no time was the server capable of decrypting anyone's data or keys. Other security features include the fact that Bob was not sent R_v until after Carol acted on recovery. The server also never sent Bob the data encrypted with k_v. The server would have canceled recovery if Carol successfully authenticated using her old credentials, thus somewhat reducing the opportunity for a malicious recovery without Carol noticing. Nonetheless, it remains possible that a malicious member of a recovery group who gains control of Carol's email could come to control Carol's 1Password account.

12.6   Recovery keys

Recovery keys are a mechanism for allowing a user to recover their account without the need for a member of the Recovery Group to be involved. This is particularly useful for accounts with a single user or in the case of a family organizer, where there may not be another user available to perform recovery using the method described above. We designed this mechanism to minimize the risk of these keys being used to enable an attacker to take over an account, thus providing greater safety than a user backing up their account password and Secret Key.

12.6.1   Recovery key generation

Recovery keys are generated by the client application using a Cryptographically Secure Pseudo-Random Number Generator (CSPRNG), with a length of 32 bytes. Following generation, the recovery key is encrypted using the user's key set symmetric key, and stored on the 1Password server. It's stored to allow the use of the key without regeneration or redistribution should the user's key set be rotated, or cryptographic components upgraded, such as the password-authenticated key exchange (PAKE) algorithm. As is the case of other secrets, such as the Secret Key, recovery keys are never exposed to the 1Password servers in unencrypted form.

Three subkeys are then generated using a hash-based key derivation function (HKDF) with the recovery key as the input keying material, and the following information values:

- 1P_RECOVERY_KEY_AUTH_v1: Authentication subkey – 32 bytes, used to authenticate the recovery key via a PAKE, to ensure the correct key is being used without exposing the key itself.

- 1P_RECOVERY_KEY_ENC_v1: Encryption subkey – 32 bytes, used to encrypt the user's key set symmetric key, allowing the user to decrypt their key set and recover their account.

- 1P_RECOVERY_KEY_UUID: Identifier - 16 bytes, used to identify the recovery key. This is a non-secret value, used by the server to identify the correct recovery key to use when multiple keys are available.

who has reestablished her account.

The hash-based key derivation function (HKDF) is used to ensure the subkeys are independent of each other, and recovery key is not exposed. Using the encryption subkey, the user's key set symmetric key is encrypted by the client application, and uploaded to the 1Password server. In addition, the client application uses Secure Remote Password (SRP), with the authentication subkey as the SRP-x, to derive a SRP-v, which is then uploaded to the server and used to authentication the recovery key to the server.

12.6.2 Recovery key authentication

When a user wants to recover their account, they must complete the following steps:

1. The client application will derive the identifier subkey from the recovery key, supplying it to the server.

2. The server will return the cryptography version number of the recovery key, and any password-authenticated key exchange (PAKE) parameters required to authenticate the recovery key.

3. The client application will derive the encryption and authentication subkeys from the recovery key, and use the authentication subkey to authenticate the recovery key to the server. This occurs with the client application using Secure Remote Password (SRP), with the authentication subkey as the SRP-x, the server will authenticate the recovery key using the previously supplied SRP-v.

Upon successful completion of these steps, the recovery key is authenticated and the client application can proceed with recovery.

12.6.3 Recovery key policies

A recovery key may implement a policy to control how it can be used. This is selected by the user when the key is generated, and stored alongside the key on the server. These policies allow additional controls to be added, beyond simple possession of the key, to ensure the user is authorized to use the key. The server won't provide the encrypted key set symmetric key to the client application until all policies are satisfied.

In addition to the polices applied as part of the recovery key itself, the server may also apply additional policies to the recovery process, this includes (at a minimum):

1. Recovery is aborted if the user successfully authentication during the recovery process.

2. Recovery is aborted if the user has successfully authenticated during the prior hour.

3. Recovery is aborted if the recovery key had an aborted attempt in the prior 24 hours.

12.6.4 Recovery key use

The server will return the encrypted key set symmetric key after authentication is complete and all recovery policies complied with. The client application can decrypt it using the encryption subkey, allowing the user to regain access to their key set. The user will then be able to regenerate their Secret Key and set a new account password, or otherwise set new root key material based on the authentication model their account uses (e.g. setting a new passkey).

12.6.5 Recovery codes

Recovery codes are an implementation of recovery keys, with a policy applied to require email verification before the recovery key can be used. As such, when a user want to recover access to their account with a recovery code, they must verify their email address, then complete the recovery process as described above.

51

If the user can't verify control of their email address, the server won't provide the encrypted key set symmetric key to the client application, and the recovery process will be aborted.

Chapter 13

Secrets Automation

As described in other sections (in particular "A modern approach to authentication" and "Transport security"), all authenticated interactions with 1Password require that the client prove knowledge of the session key without revealing any secrets. That session key in turn can only be established through proof of knowledge or access to the account password and Secret Key.

This aspect of our security design makes it much harder for someone to work their way around 1Password's authentication, as every request to the service is cryptographically bound to the authentication process itself. It also limits the number of authentication attempts a client can perform in a particular time period.

This security design introduces a challenge when automated processes need to retrieve, modify, or create secrets in 1Password. Such apps and processes are not designed to sign in to 1Password directly; typically those processes are designed to authentication through more traditional means. Reauthenticating for each request would be cumbersome at best.
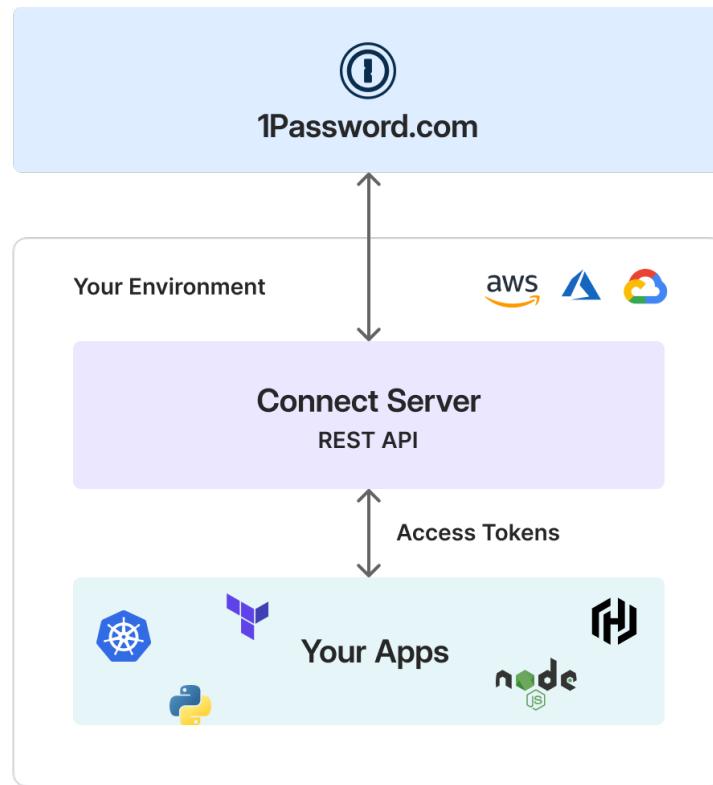
Figure 13.1: *The 1Password Connect server lives in your environment and acts as a RESTful connection between your apps and the 1Password service.*

The overall solution we provide as part of Secrets Automation is a Connect Server. It's capable of signing in to 1Password directly, and apps and automated processes can interact with it through a RESTful interface. The API for the Connect Server can be called by customer-created clients or built with plug-ins we offer.

Any automated process given power over an organization's resources, particularly the kinds of resources managed within 1Password, creates an area of attack.[19] Therefore, it's necessary to design them with security in mind. In general, there are two principles to adhere to when deploying automations like these.

- The power allotted to any given automation should be closely tied and limited to what it's expected to do.

- The credentials required by the automation to perform its duties must be securely managed.

1Password Secrets Automation is designed according to these principles — it's also design to help customers *follow* the same principles.

You may want to read through what follows multiple times,[20] because many of the interacting parts are mentioned before they're fully defined. In particular, the descriptions of the credentials JSON and bearer token each depend on each other.

13.1   The Connect server

The Connect server is deployed in your environment and serves as a bridge between the client processes and the 1Password service. Although simple in principle, there are a number of interacting parts, so it's useful to start with an overview and quick facts about it that will be elaborated on later in this chapter.

- The 1Password Connect server has encrypted credentials necessary to sign in to 1Password under a specific service account.

- The service accounts used by Connect servers are not given any ability to manage users or vaults.

- The Connect server is deployed by the user in the user's environment. AgileBits has no means of accessing it.

- Authentication to the Connect server is through use of an HTTP Authorization header bearer token.

- The credentials necessary to sign in as the service account are split between what's stored with the Connect server and bearer token.

13.1.1   Service account

1Password service accounts aren't highly visible to users, but it's important to mention them to better describe how Secrets Automation works in practice.

Generally speaking, a service account is a special user within a 1Password account, but the user isn't associated with a person or group of people. They help an organization manage the secrets used by entities with very specific roles and functions.

If Patty, a member of a 1Password account, is responsible for ensuring backups can be restored, they may need access to the credentials for the backups system. But the automated processes that perform backups and restorations shouldn't have all of Patty's 1Password privileges. They should only have the privileges required to perform their duties.

The appropriate service accounts are set up in 1Password when setting up Secrets Automation. During creation of a service account, the administrator will select which vaults the service account will have access to and share those vaults with the service account. In this respect, the service

---

[19]This is true whether or not those processes are systematically managed within an organization or are left untracked. It is not difficult to guess which might introduce more risk.

[20]Or, perhaps, zero times.

account is like an ordinary user. Unlike an ordinary user, the service account has no management privileges and will be prevented from further sharing the vaults to which it has been given the keys by the 1Password backend system.

The account password for the service account is randomly generated and discarded after deriving the Account Unlock Key (AUK) and SRP-x. Key generation is performed client side, either in the web client or the command-line interface (CLI).

> **Dangerous bend**
>
> The service accounts created for Secrets Automation complement the ones created for use by the SCIM bridge, and used for automated management of 1Password users. They have the ability to create and delete users, and add users to groups, but no ability to retrieve data from vaults.

13.1.2 Local deployment

The Connect server, with the encrypted 1Password credentials, is deployed on your system. At initial release, April 2021, we provided setups for deploying it within a Docker container or via Kubernetes. At no time does AgileBits have access to the Connect server or data it stores.

13.1.3 Credential store

The Connect server locally stores encrypted credentials, containing the Account Unlock Key (AUK) and SRP-x for the associated service account. This saves it from having to go through the entire key derivation process each time it needs to start a 1Password session.

```
{ "verifier": { ... },
"encCredentials":
    { ... },
"uniqueKey": { ... },
"version": "2",
"deviceUuid": "22e ..." }
```

Figure 13.2: *An overview of the credentials file, with three major components and some header information.*

13.1.4 The credentials file

When setting up a Connect server initially, the user's 1Password client constructs a `1password-credentials.json` file along with a bearer token. The credentials file has three substantive components: The `verifier` is used as part of an additional authentication of the bearer token; `encCredentials` contain the encrypted credentials necessary for the associated service account to sign in to 1Password; and `uniqueKey` is key shared between the client-facing Connect server and the Connect server syncher.

13.1.5 Encrypted credentials

The encrypted credentials, illustrated in Figure 13.3 contain, unsurprisingly, the encrypted 1Password credentials required to unlock 1Password as the associated service account.

```
"encCredentials": {
     "kid": "localauthv2keykid",
     "enc": "A256GCM",
     "cty": "b5+jwk+json",
     "iv": "VSu ...",
     "data": "AZ1H0WprT ..."}
```

Figure 13.3: *The 'encCredentials' object is a JSON web key (JWK) used to encrypted 1Password unlocking credentials. It's encrypted with a key derived from the bearer token.*

Encryption, as with all our symmetric encryption, is with Advanced Encryption Standard (AES) using Galois Counter Mode (GCM) for authenticated encryption. The nonce is given in the iv.

When decrypted, the object is structured as in the Golang structure in Figure 13.4. The URL will typically be something like example.1password.com; the email addresses created for service accounts are never expected to be used for email, and only serve as a username. The user Universally Unique Identifier (UUID) uniquely identifies the service account Secret Key[21], SRP-x, and Account Unlock Key (AUK) are as described in "A deeper look at keys."

```
type SigninSRPComputedXCredentials struct {
     URL            string
     Email          string
     UserUUID       string
     SecretKey      *crypto.SecretKey
     SRPComputedX   *crypto.SRPComputedX
     HexMUK         string
}
```

Figure 13.4: *Decrypted credential structure. The URL, Email, UserUUID, and SecretKey are used to identify the user, account, and service. The SRP-x and AUK are the secrets required to authenticate with 1Password and decrypt (the keys needed to decrypt the keys which encrypt) the vault data.*

```
{ "verifier": {
     "salt": "Em ...",
     "localHash": "Yvra ..." }
```

Figure 13.5: *Connect server verifier*

---

[21] It's only the non-secret part of the Secret Key used in the process. All service account identifying information must be consistent for successful authentication.

13.1.6 Verifier

The token within the bearer token is run through a key derivation function, which must match the verifier stored by the Connect server.

This verification is redundant, as the signature verification of the entire bearer token provides all the authentication necessary and guarantees the integrity of the request.

13.1.7 Interprocess key

The 1Password Connect server has two running processes. One provides the user-facing service while the other synchronizes data with 1Password itself. Among other things, this allows the Connect server to operate even when a direct connection to 1Password is unavailable.[22] This also allows for much faster responses from the Connect server. The data stored by the sync server is encrypted as with any 1Password client.

```
"uniqueKey": {
    alg": "A256GCM",
    "ext": true,
    "k": "DCpU ...",
    "key_ops": ["decrypt", "encrypt"],
    "kty": "oct",
    "kid": "lyt ..." }
```

Figure 13.6: *Connect server IPC key. The Connect server interprocess key is used to secure communication between the sync server and the client facing Connect service.*

The interprocess key, here called `uniqueKey`[23] is used as a shared secret between the client-facing Connect server and synchronization server in order to encrypt the bearer token between them.

13.2 Bearer token

The bearer token is a JSON web token (JWT) that's transmitted from the user's client process to the Connect server using the HTTP Authorization header. It contains a key that's used, indirectly, to decrypt the 1Password credentials stored by the Connect server. It also contains claims, in the JWT sense, listing what 1Password vaults it has access to. As a signed JWT, it's also used directly for authentication to the Connect server. Serialized JWTs are composed of three base64-encoded parts: header, payload, and signature. These parts are separated by the "." character.

---

[22]This might be particularly handy if you are managing your network equipment with Secrets Automation.

[23]All keys are unique, but are some keys more unique than others? They're all unique, but coming up with names for yet another key when developing something is difficult, and the temporary placeholder name may stick around longer than anyone might expect.

Figure 13.7: *Sample JWT header for bearer tokens*

13.3 Header

An example header portion is shown in Figure 13.7. The `kid` identifies the signing key of the corresponding service account key set, which is used to sign the bearer token. It must be a key belonging to the subject field in the payload.

> **Dangerous bend**
>
> Although Elliptic Curve Digital Signature Algorithm (ECDSA) isn't the most robust of digital signature algorithms, it's the one we settled on for the timebeing, as it's widely available in well-vetted cryptographic libraries. We find it particularly important to whitelist the algorithms that we accept in a JWT, because it helps avoid a number of security concerns[a] surrounding JSON Object Signing and Encryption (JOSE) and JWT.
>
> In particular, the flexibility of signature and encryption algorithms can lead to downgrade attacks. In addition to whitelisting signature algorithms (currently ECDSA using P-256 and SHA-256 (ES256) is the only one the Connect server will accept) our verification process is very aggressive in rejecting inconsistent or malformed tokens.
>
> ---
> [a]@Arciszewski17:JOSE

13.3.1 Payload

A sample payload, or claims, portion of the bearer token can be seen in Figure 13.8.

```
{ "1password.com/auuid":"XRS ...",
  "1password.com/token":"pMb ...",
  "1password.com/fts": ["vaultaccess"],
  "1password.com/vts":[{
    "u":"57o...", "a":48}],
  "aud":["com.1password.connect"],
  "sub":"6YUT ...",
  "exp":1625961599, "iat":1618167578,
  "iss":"com.1password.b5",
  "jti":"j24 ..." }
```

Figure 13.8: *Sample JWT payload for bearer tokens. 'AUUID' is the account UUID, and the subject 'sub' is the user UUID for the service account user. The features, 'fts', will always be 'vaultaccess' for Secrets Automation. The 'token' is both an authentication secret to the Connect server and key used to derive the key to decrypt the 1Password credentials stored on the Connect server.*

Most of what appears in the figure can be understood from the JSON web token (JWT) standards, which you may peruse at your leisure. What requires explanation follows.

- `sub` The subject of the bearer token is the UUID of the service account that signs into 1Password.

- `auuid` The account UUID.

- `fts` Features will always be "`vaultaccess`" for Secrets Automation.

- `vts` The vaults the client is claiming access to, along with its read and write claims.

- `token` The token which, among other things, is used to decrypt credentials stored by the Connect server.

It's worth noting that a particular service account may have more access to more vaults than claimed in the bearer token payload. The Connect server won't honor client requests that go beyond the validated claims.

For example, if the associated service account has the ability to read and write to vaults V_1 and V_2, while the signed claim is only for reading V_1, the Connect server will only honor read requests for V_1. Naturally, if the service account associated with one of these tokens doesn't have any access to V_3 but somehow shows up with a valid claim to it, the Connect server will reject the claim.

Even if the Connect server were somehow tricked into honoring such a claim, the 1Password service wouldn't return the data, and the Connect server wouldn't be able to decrypt the data even if it were returned.

13.3.2 Signature

The third part of the bearer token is the JSON web token (JWT) signature. The signature is created by the associated service account using that account's signing key. This signature covers the payload of the bearer token, preventing tampering or forgery.

Chapter 14

Transport security

We designed 1Password with the understanding that data traveling over a network can be read and tampered with unless otherwise protected. Here we discuss the multiple layers of protections we have in place. Roughly speaking, there are three layers of protection.

1. 1Password's **at-rest** encryption, as described in "How vault items are secured," also applies to data when it's in transit.

Your items are always encrypted with vault keys, which in turn are encrypted by keys held by you and not by the server. They remain encrypted this way in transit.

2. Transport Layer Security (TLS) with best practices (encryption, data integrity, authenticity of server).

TLS the successor of SSL, puts the "S" in "HTTPS." It encrypts data in transit and authenticates the server so the client knows to whom it's talking.

3. Secure Remote Password (SRP) authentication and encryption

The login process provides mutual authentication. Not only does your client prove who it is to the server, but the server proves who it is to the client. This is in addition to the server authentication provided by TLS. During login, a session key will be agreed upon between client and server, and communication will be encrypted using Advanced Encryption Standard (AES) in Galois Counter Mode (GCM).

The protocol provides a layer of authentication and encryption that's independent of Transport Layer Security (TLS).

When discussing transport security, it's useful to distinguish different security notions: **integrity**, **authenticity**, and **confidentiality**.[24] "Confidentiality" means the data remains secret, "authenticity" means the parties in the data exchange are talking to whom they believe they're talking to, and data "integrity" means the data transmitted can't be tampered with. Tampering includes not only changing the contents of a particular message, but also preventing a message from getting to the recipient or injecting a message into the conversation the authorized sender never sent.

Because parts of systems can fail, it's useful to design the overall system so a failure in one part doesn't result in total failure. This approach is often called defense in depth.

As summarized in Table 14.1, each encryption layer is independent of the others. If one fails, the others remain in place (though see A.1 for an exception). The at-rest encryption described in "How vault items are secured is not part of a communication protocol, and so authentication is not applicable to it. Transport Layer Security (TLS), as it's typically used, authentication the server but doesn't authenticate the client.

---

[24]When discussing information security, the acronym "CIA" is often used to refer to confidentiality, integrity, and *availability*. But when considering data transport security, integrity and authenticity play a major role. In neither case should the abbreviation be confused with the well-known institution, the Culinary Institute of America.

Table 14.1: *All these mechanisms are used to protect data in transit. "SRP+GCM" refers to the combination of SRP and our communication encryption; "at-rest encryption" refers to the normal encryption when stored.*

	SRP+GCM	TLS	AT-REST ENCRYPTION
Confidentiality	✓	✓	✓
Data integrity	✓	✓	✓
Server authenticity	✓	✓	✗
Client authenticity	✓	✗	✗

One limitation of SRP+GCM is that each message is encrypted individually. An attacker who can get in the middle of that connection, could replay messages sent over SRP+GCM and the server will accept them. We'd like to expand the security goals of this transport encryption such that messages cannot be replayed in the future.

14.1   Data at rest

Your 1Password data is always encrypted when it's stored anywhere[25] whether on your computer or on our servers, and it's encrypted with keys that are encrypted with keys derived from your account password and Secret Key. Even if there were no other mechanisms to provide data confidentiality and integrity for the data that reaches the recipient, 1Password's at-rest encryption sufficiently provides both.

Because it's designed for stored data, this layer of data encryption doesn't ensure messages can't go missing or older data is not replayed. It also doesn't authentication the communication channel.

14.2   TLS

Transport Layer Security (TLS) puts the "S" in "HTTPS". It provides encryption, data integrity, and authenticity of the server.

Our TLS configuration includes HTTP Strict Transport Security (HSTS) and a restricted set of cipher suites to avoid downgrade attacks. Precise policies and choices will change more rapidly than the document you're reading will be updated.

Neither certificate pinning nor DNSSec have been implemented. Given the mutual authentication described in "A modern approach to authentication," the marginal gain in security provided by such measures isn't something we consider to be worth the risk of availability loss should those extra measures fail in some way. Following research[26] and analysis[27] of the value of certain security indicators and extended validation certificates in particular, we're no longer using extended validation certificates.

14.3   Our transport security

Our use of Secure Remote Password (SRP) authentication between the client and server provides mutual authentication. Both the server and client will know they're talking to exactly who they think they're talking to.

---

[25] Decrypted Documents may be written to your device's disk temporarily after you open them.
[26] Jackson et al. (2007)
[27] Hunt (2019)

This is in addition to the server authentication provided by Transport Layer Security (TLS). Thus, if TLS fails in some instances to provide proper authentication, SRP still provides authentication.

Not only does the client prove its identity to the server, but the server proves its identity to the client.

14.3.1   Client delivery

This section has focused on the transport security between 1Password clients and server. For discussion of delivery of the client itself see A.1 in "Beware of the leopard."

14.3.2   Passkey and single sign-on unlock caveats

You can use a passkey or single sign-on (SSO) to unlock a 1Password account, as described in @ref(#passkeySSO). When you sign in with a passkey, that sign-in with the 1Password server is only protected by Transport Layer Security (TLS). When you sign in with your SSO provider, they're responsible for protecting your sign-in information on the network. Single sign-on providers generally only protect the confidentiality of login information using TLS.

After completing authentication with either method, a client will fetch an encrypted credential bundle from the server. A client can only use Secure Remote Password (SRP) **after** fetching this bundle. If an attacker can break the security of Transport Layer Security (TLS), they can obtain an encrypted copy of the credential bundle.

Chapter 15

Server infrastructure

15.1   What the server stores

1Password stores account, team, vault, and user information in a relational database. Membership in teams and access to team resources, including vaults, groups, and items, are determined by fields within each database table. For example, the users table includes three fields used to determine user identity and team membership. These fields are uuid, id, and account_id. The user's account_id field references the accounts table id field, and this relationship determines membership within an account.

These relationships — users to accounts, accounts to vaults, vaults to items — don't determine a user's ability to encrypt or decrypt an item, they only determine the ability to access the records. The relationship from a user to an item within a team vault is as follows:

- A users table entry has an account_id field that references the id field in the accounts table.

- An accounts table entry has an id field which is referenced by the account_id field in the vaults table.

- A vaults table entry has an id field which is referenced by vault_id field in the vault_items table.

- A vault_items table entry has the encrypted_by, enc_overview, and enc_details fields which reference the required encryption key and contain the encrypted overview and detail information for an item.

A malicious database administrator may modify the relationships between users, accounts, teams, vaults, and items, but the cryptography will prevent the items from being revealed.

Principle 3 states the system must be designed for people's behavior, and that includes malicious behavior. A malicious database administrator may be able to modify the relationships between users and items, but he will be thwarted by the cryptography when he, or his cohort in crime, attempts to decrypt the item. The cryptographic relationship between a user and an item within a team vault is as follows:

- A vault_items entry has a vault_id field which references the vault_id field in the user_vault_access table. The enc_overview and enc_details fields in a vault_items entry are encrypted with the key contained in the enc_vault_key field of the corresponding user_vault_access entry, which is encrypted itself.

- A user_vault_access entry is located using the id field for the users table entry and id field for the vaults table entry. The enc_vault_key field in the user_vault_access entry is encrypted with the user's public key and may only be decrypted with the user's private key.

- A users entry is located using the email address the user provided when signing in and the accounts entry for the matching domain. The users entry includes the pub_key field which is used to encrypt all the user's secrets.

With the hard work of the malicious database administrator, the user may have access to a user_vault_access table entry which has the correct references, but since 1Password never has a copy of the unencrypted vault key, it's impossible for the user to have a copy of the vault key

encrypted with her public key. The malicious database administrator could copy the encrypted vault key for another user, but the user wouldn't have the private key required to decrypt the encrypted vault key.

Principle 2 states we should trust the math, and as has been shown here, even if a malicious database administrator were to modify the account information to grant a user access to an encrypted item, the user still lacks the secrets needed for decryption. The attacker has been foiled again.

Finally, principle 4 states that our design should be open for review. While we hope our database administrators don't become malicious, we've provided all the information needed to grant unauthorized access to encrypted items knowing they will remain protected by cryptography.

The example of a malicious database administrator was chosen because the worst-case scenario is someone sitting at a terminal on the back end server, issuing commands directly to the database, with a list of database tables and column names in hand.

15.2   How your data is stored

1Password stores all database information using an Amazon Web Services Aurora database instance. The Amazon Aurora service provides a MySQL-compatible SQL relational database. Aurora provides distributed, redundant and scalable access. Some of the tables and their uses were provided earlier.

Data is organized in the traditional manner for a relational database, with tables consisting of rows and columns,[28] with various indices defined to improve performance.

Binary data, which may include compressed JSON objects representing key sets, templates, and other large items is compressed using ZLIB compression as described in RFC 1950.

The tables are listed as follows:

- `accounts` Contains registered teams, which originated from an initial signup request, approval, and registration. This table includes the cleartext team domain name (`domain`), team name (`team`) and avatar (`avatar`). Other tables will typically reference the `accounts` table using the `id` field.

- `groups` Used to reference groups of users in a team. The `groups` table is primarily referenced by the `group_membership` and `group_vault_access` tables. This table includes the cleartext group name (`group_name`) and description (`group_desc`), public key (`pub_key`), and avatar (`avatar`).

- `invites` Contains user invitations. The unencrypted `acceptance_token` is used to prevent inappropriate responses to an invitation and not relevant once a `user` has been fully initialized. The remaining unencrypted columns are the user's given name(`first_name`), family name (`last_name`) and email address (`email`).

- `signups` Contains user requests to use the 1Password server. This table includes the cleartext team name (`name`) and email address of the requester (`email`).

- `users` Contains registered users, which originated via the invitation process and were eventually confirmed as users. This table includes the cleartext user name (`first_name` and `last_name`), email address (`email`), a truncated copy of the lower-case email address (`lowercase_email`), the user's public key (`pub_key`), and an avatar (`avatar`).

Aggregating the list of unencrypted fields above, the data subject to disclosure in the event of a data breach or required disclosure are:

---

[28]Only the cleartext columns will be listed at present as these are the columns which would be disclosed in the event of a data breach. The encrypted columns will be protected by the security of the various keys which the server doesn't possess.

- Team domain name, long-form name, and avatars.

- IP addresses used by devices

- MFA secrets

- Client device makes, models, operating systems, and versions

- Public keys, which are intended to be public.

- Group names, descriptions, and avatar file names.

- Users' full names, email addresses, and avatar file names.

Appendix A

Appendix A: Beware of the leopard

> **Beware of the leopard**
>
> "You hadn't exactly gone out of your way to call attention to them had you? I mean like actually telling anyone or anything."
> "But the plans were on display…"
> "On display? I eventually had to go down to the cellar to find them."
> "That's the display department."
> "With a torch."
> "Ah, well the lights had probably gone."
> "So had the stairs."
> "But look you found the notice didn't you?"
> "Yes," said Arthur, "yes, I did. It was on display in the bottom of a locked filing cabinet stuck in a disused lavatory with a sign on the door saying *Beware of the leopard*."[a]
>
> ---
> [a]@hhg79

This chapter discusses places where the actual security properties of 1Password may not meet user expectations.

A.1 Crypto over HTTPS

1Password offers a web client which provides the same end-to-end (E2E) encryption as when using the native clients. The web client is fetched from our servers as a set of JavaScript files (compiled from TypeScript source) that's run and executed locally in the user's browser on their own machine. Although it may appear to users of the web client that our server has the capacity to decrypt user data, all encryption occurs on the user's machine using keys derived from their account password and Secret Key. Likewise authentication in the web-client involves the same zero-knowledge authentication scheme described in 4.

Despite that preservation of end-to-end (E2E) encryption and zero-knowledge authentication, the use and availability of the web client introduces a number of significant risks.

- **The authenticity and integrity of the web client depends on the integrity of the TLS connection by which it's delivered.** An attacker capable of tampering with the traffic that delivers the web client could deliver a malicious client to the user.

- **The authenticity and integrity of the web client depends on the security of the host from which it's delivered.** An attacker capable of changing the web client on the server could deliver a malicious client to the user.

- **The web client runs in a very hostile environment: the web browser.** Some attacks on the browser (like a malicious extension) may be able to capture user secrets. This is discussed further in A.1.1.

- **Without the web-client users would only enter their account password into native clients and so would be less vulnerable to phishing attacks.**

- **The web client creates the false impression for many users that encryption is not end-to-end.** Although this may not have direct security consequences for the user, it may re-enforce unfortunately low expectations of security in general.

User mitigations include:

- Use (code signed) native clients as much as possible.

- Keep browser software up to date

- Create a specific browser profile for using the web-client

- Pay close attention to browser security warnings

- Use only on trusted networks.

- Manually check certificates

Our mitigations include:

- Use the most recent Transport Layer Security (TLS) version

- Don't support weak cipher suites (so avoiding many downgrade attacks)

- Use of safe JavaScript constructions.

- Use HTTP Strict Transport Security (HSTS) (so avoiding HTTPS to HTTP downgrade attacks)

- Pin Certificates (not yet implemented)

> **Browser warnings**
>
> Always be sure to heed all browser warnings regarding TLS connections.

A.1.1 Crypto in the browser

Running security tools within a browser environment brings its own perils, irrespective of whether it's delivered over the web. These perils include:

- The browser itself is a hostile environment, running processes and content that are neither under your control nor ours. Sandboxing within the browser provides the first line of defense. Structuring our in-browser code to expose only what needs to be exposed is another. Over the past decade, browsers have made enormous improvements in their security and in their isolation of processes, but it still remains a tough environment.

- JavaScript, the language used within the browser, offers us very limited ability to clear data from memory. Secrets we'd like the client to forget may remain in memory longer than useful.

- We have a strictly limited ability to use security features of the operating system when operating within the browser. See section A.10.2 for how this limits the tools available for protecting the Secret Key when stored locally.

- There's a paucity of efficient cryptographic functions available to run in JavaScript. As a consequence, the WebCrypto facilities available in the browsers we support impose a limit on the cryptographic methods we can use. For example, our reliance on PBKDF2 instead of a memory-hard KDF such as Argon2 is a consequence of this.

A.2   Recovery Group powers

From a cryptographic point of view, the members of a Recovery Group have access to all the vault keys in that group.[29] Server policy restricts what a member of the Recovery Group can do with that access, but if a Recovery Group member is able to defeat or evade server policy and gain access to an encrypted vault (for example, as cached on someone else's device) then that Recovery Group member can decrypt the contents of that vault.

Depending on the nature of the threat to the team's data and resources an attacker will put into acquiring it, members of the Recovery Group and their computers may be subject to targeted attacks.

> **Recovery group members**
>
> Members of the recovery group must be selected with care and keep their systems secure.

A.3   No public key verification

At present there's no practical method[30] for a user to verify the public key they're encrypting data to belongs to their intended recipient. As a consequence it would be possible for a malicious or compromised 1Password server to provide dishonest public keys to the user, and run a successful Man in the Middle (MITM) attack. Under such an attack, it would be possible for the 1Password server to acquire vault encryption keys with little ability for users to detect or prevent it

This is discussed in greater detail in "Appendix C."

A.4   Limited re-encryption secrecy

A.4.1   Revocation

Removing someone from a vault, group, or team isn't cryptographically enforced. Cryptographic keys are not changed.

A member of a vault has access to the vault key, as a copy of the vault key is encrypted with that member's public key. When someone is removed from a vault, that copy of the vault key is removed from the server, and the server will no longer allow that member to get a copy of the vault data.

If prior to being removed from a vault the person makes a copy of the vault key which they store locally, they will be able to decrypt all future data if they find a way to obtain the encrypted vault data. This is illustrated in Story 10. Note this requires the attacker both plan ahead and somehow acquire updated data.

---

[29]1Password Teams accounts also have a permission called "Manage All Groups" with equivalent cryptographic access, which is only given to the Administrators and Owners groups by default.

[30]An impractical method for the users to run 1Password in a debugger to inspect the crucial values of the public keys themselves. Additionally, the 1Password command line utility (as of version 0.21), has an undocumented method to display public keys and fingerprints of users.

> **Story 10: Mr. Talk is not a good team player**
>
> **[Monday]** Patty (a dog and Team administrator) adds Mr. Talk (neighbor's cat) to the Squirrel Watchers vault. Molly (another dog) is already a member.
> **[Tuesday]** Mr. Talk makes a copy of all of his keys and stores that copy separately from 1Password.
> **[Wednesday]** Mr. Talk is discovered stealing Patty's toys and is expelled from the vault (and from the team).
> **[Thursday]** Patty updates the Squirrel Watchers vault with the new hiding place for her toys.
> **[Friday]** Mr. Talk manages to steal a cached copy of the encrypted vault from Molly's poorly secured device. (Molly still hasn't learned the importance of using a device passcode on her phone.)
> **[Saturday]** Mr. Talk decrypts the data he stole on Friday using the keys he saved on Tuesday, and is able to see the hiding place Patty added on Thursday.
> To launch the attack, Mr. Talk needed to acquire a copy of the encrypted data the server would no longer provide, and he needed to anticipate being fired.

A.4.2   Your mitigations

If you feel that someone removed from a vault may have a store of their vault keys and will somehow be able to acquire new encrypted vault data despite being denied access by server policy, then it's possible to create a new vault (which will have a new key), and move items from the old vault to the new one. Someone revoked from a vault won't be able to decrypt the data in the new vault no matter what encrypted data they gain access to.

A.5   Account password changes don't change keysets

A change of account password or Secret Key does not create a new personal keyset, it only changes the Account Unlock Key (AUK) with which the personal key set is encrypted. Thus an attacker who gains access to a victim's old personal key set can decrypt it with an old account password and old Secret Key, and use that to decrypt data that was created by the victim after the change of the account password.

A.5.1   Your mitigations

A user's personal keyset may be replaced by voluntarily requesting their account be recovered. This will create a new personal keyset which will be used to re-encrypt all the vault keys and other items which were encrypted with the previous personal keyset.

A.6   Local client account password has control of other account passwords

Most 1Password client applications can handle multiple 1Password user accounts. It's common, perhaps even typical, for an individual to have a 1Password membership as part of the business or organization they're a member of, as well as being a member of their own 1Password family.

Most 1Password clients are designed to unlock all accounts when unlocked. The account that will locally contain the encrypted secrets to unlock the others is called the primary account. It's (for most clients) the first account the client signed into. The precise details of how this is handled can vary from client to client, but in essence, the secrets needed to unlock a secondary account

(the Account Unlock Key (AUK) and SRP-x) are encrypted with (keys encrypted by) the AUK of the primary account.

The security risk is that account password policies that may be set and expected by an organization won't be followed in practice if the account with such policies is a secondary account for a particular client.

> **Story 11: A weak primary account password unlocks a stronger account**
>
> Molly (a dog) is a member of a business account for Rabbit Chasers Inc., and Patty, an administrator for Rabbit Chasers Inc., has used the features of a business account to set very strict account password requirements for all of its members. So Molly's account password for that account does conform to that account's requirements. Patty is naturally under the impression that Molly must use the strong account password when unlocking her work account.
>
> But Molly is also a member of a family account, and in her family account she has set her password to be `squirrelrabbit`, which is easily guessable by anyone familiar with Molly. Furthermore, Molly set up her family account first when she set up 1Password on her device. She added her work account later.
>
> When she first added her work account to that device, she had to enter the strong account password for that account, but every time she unlocks 1Password thereafter, she unlocks both accounts with `squirrelrabbit`.
>
> One day the evil neighborhood cat, Mr. Talk, steals Molly's device. Mr. Talk can guess Molly's weak family account account password, and unlocking 1Password on Molly's computer can now unlock Molly's work account as well.
>
> Patty is not amused.

An additional problem with this scheme is that users are more likely to forget they have a separate account password for their secondary account(s), and are more likely to forget those passwords.

A.6.1   Mitigations

There are no mitigations for users of 1Password 7 and earlier other than risk awareness. 1Password 8 periodically requests the user's account password by default.

A.7   Policy enforcement mechanisms not always clear to user

Readers of this document may recall from "Access control enforcement" that some controls (such as the ability to decrypt and read the contents of a vault) are enforced through cryptography, while others are enforced only through the client user interface (such as the ability to print the contents of a vault they have use access to). The security properties of those differ enormously. In particular, it's very easy to evade policy that is only enforced by the client.

Many team administrators will not have read this document or other places where the distinction is documented. Therefore, there's a potential for them to have an incorrect impression of the security consequences of their decisions.

A.8   Malicious client

There's no technical barrier to a malicious client, which might generate bad keys or send keys to some third party.

A.9   Vulnerability of server data

It should be assumed that governments, whether through law enforcement demands or other means, may gain access to all the data we have or our data hosting provider has. This may happen with or without our knowledge or consent. The same is true for non-governmental entities which may somehow obtain server data. Your protection is to have a good account password and to keep your Secret Key secure.

Although we may resist law enforcement requests, we obey the laws of the jurisdictions in which we are obliged to do so.

A.10   Malicious processes on your devices

Malware that can inspect the memory of your computer or device when the 1Password client has unlocked your data will be able to extract secrets from the 1Password process. Malware that can install a malicious version of 1Password and have you execute it will also be able to extract your secrets. After malware running on a system has gained sufficient power, there's no way in principle to protect other processes running on that system.

But we must also consider the threat posed by less powerful malware, and in particular with respect to the exposure of the Secret Key.

A.10.1   Malicious or undesired browser components

When you use 1Password in your web browser, browser extensions – even built-in browser features – can expose the data you fill into your browser. This can have explicit malicious intent, like when a browser extension monitors the data input into text fields to spy on you. Sometimes this can be accidental, such as when browser extensions submit the data you put into text fields to perform autocompletion features, perform translation, or store or analyze the text you're typing in some other way.

The 1Password browser extension tries to avoid filling certain form fields if it suspects data may be submitted elsewhere; however, you should use caution when selecting your web browser and extensions, and attempt to understand if and when they send text you enter other places.

A.10.2   Locally exposed Secret Keys

After a client is enrolled, it will store a copy of the Secret Key on the local device. Because the Secret Key must be used to derive the user's Account Unlock Key (AUK) it cannot be encrypted by the same AUK or by any key directly or indirectly encrypted with the AUK. Depending on client and client platform, the Secret Key may[31] be stored on the device using some of the protections offered by the operating system and lightly obfuscated. But it should be assumed that an attacker who gains read access to the user's disk will acquire the Secret Key.

Recall from the discussion of two-secret key derivation (2SKD) in 4.1.3 that the Secret Key is designed so an attacker won't be in a position to launch an offline password-guessing attack if they capture data from our server alone. That is, the Secret Key provides extremely strong protection for users if **our** servers were to be breached. The Secret Key plays no security role if the **user's** system is breached. In the latter situation, the strength of the user's account password[32] determines whether an attacker will be able to decrypt data captured from the user's device.

---

[31]We're deliberately vague about this, as practice may change rapidly from version to version, including different behaviors on different operating system versions.

[32]The slow hashing (described in 8.2.4) in our key derivation function goes some way to increase the work that an attacker must do to verify account password guesses from data captured from the user, but it cannot substitute a strong account password.

A.10.3   Device keys used with passkey and single sign-on unlock

A client enrolled using passkey or single sign-on (SSO) unlock (described in "Unlock with a passkey or SSO") doesn't store a Secret Key locally. Instead it stores a device key locally. The combination of a device key and successful passkey or SSO authentication is required to unlock a 1Password account on those devices.

On iOS, macOS, and Android devices, we protect the device key with the device's hardware security features; other devices don't reliably allow protecting an encryption key with hardware, nor do web browsers on any platform. In cases which we can't store the device key protected by a hardware mechanism, we store it (lightly obfuscated) on the computer's storage drive. This means malware could read the device key and can use it to attempt to access a user's 1Password account.

If Oscar runs malware on Alice's computer when she uses single sign-on (SSO) to unlock 1Password, he can steal both Alice's device key and Alice's SSO session cookies from her browser's cache. Oscar can use the combination of items to unlock Alice's 1Password account. Similarly, if Oscar has access to the hard drive contents from Alice's computer (like when he gains access to a backup of Alice's computer or performs forensic analysis on her computer) he can copy this information and unlock Alice's 1Password account, as well.

Devices that unlock with passkey store their passkey unlocking information in their operating system's passkey provider. We rely on the operating system and device manufacturer to prevent malware from being able to steal the authentiction information for passkeys. We don't use session information stored in the web browser to unlock accounts with passkeys.

Because of the risks of the device key and single sign-on authorization data sitting together on a disk we only offer single sign-on (SSO) to businesses that can weigh the risks and rewards of using SSO with device security aspects. Businesses using SSO can configure the details of devices used, the single sign-on provider used, and the way single sign-on is used within 1Password to fit their individual security needs.

A.11   Revealing who is registered

If Oscar suspects that `alice@company.example` is a registered user in a particular Team or Family, it's possible for him to submit requests to our server that would allow him to confirm an email address is or isn't a member of a team. Note that this does **not** provide a mechanism for enumerating registered users, it's only a mechanism that confirms whether or not a particular user is registered. Oscar must first make his guess and test that guess.

We attempted to prevent this leak of information and believed we had. A design error (that's difficult to fix) means we must withdraw our claim of that protection.

A.12   Use of email

Both invitations and recovery messages are sent by email. It's very important that when administrators or Recovery Group member take actions that result in sensitive email being sent, they check with the recipients through means other than email that the messages were received and acted upon.

Appendix B

Appendix B: Secure Remote Password

In "A modern approach to authentication," we spoke of mathematical magic.

> Using some mathematical magic the server and the client are able to send each other puzzles that can only be solved with knowledge of the appropriate secrets, but no secrets are transmitted during this exchange.

This appendix offers some insight into that magic.

We insist on this magic even though 1Password's principle source of security is through end-to-end (E2E) encryption instead of authentication. We need to ensure our authentication system would never provide an attacker the means to learn anything about the secrets needed to decrypt someone's data — even if it were compromised.

We use Secure Remote Password (SRP) as our password-authenticated key exchange (PAKE) to achieve the authentication goals set out in Figure 4.1. With SRP, the client can compute from a password (and a few other things) a number that is imaginatively called x. This secret x is never transmitted.

B.0.1 Registration

The client computes x from the user's account password and Secret Key and from some non-secret information as described in section B.0.3.

During first registration, the client will compute from x a verifier, v. During initial registration, the client sends v to the server, along with a non-secret salt. The client and the server also need to agree on some other non-secret parameters. The verifier is the only sensitive information ever transmitted, and it's sent only during initial registration.

Dangerous bend

The verifier v cannot be used directly to compute either x or the password that was used to generate x; however, it's similar to a password hash in that it can be used in password cracking attempts. That is, an attacker who has acquired v can make a password guess and see if processing that guess yields an x that produces the v. Our use of two-secret key derivation (2SKD) makes it impossible to launch such a cracking attack without also having the user's Secret Key.

B.0.2 Sign-in

The client will be able to compute x from the account password, Secret Key, and salt as described in 8.2.[33] The server has v. Because of the special relationship between x and v, the server and client can present each other mathematical challenges that achieve the following:

- Prove to the server the client has the x associated with v.
- Prove to the client the server has the v associated with x.

---

[33]The client may locally store x in a way that's encrypted with keys that depend on the Account Unlock Key (AUK) instead of recalculating it afresh each time.

- Let the client and server agree on a key S which can be used for further encrypting the session.

During that exchange, no information about the user's password is revealed, nor is any information about x or v or S revealed to someone listening in. Furthermore, the mathematical challenge the client and server present to each other is different each time, so one session cannot be replayed.

B.0.3   With a strong KDF

The standards documents describing Secure Remote Password (SRP) offer the generation of x from the password, P; salt, s; and username, I; as in Figure B.1. The values of g and N are public parameters that will be further described in B.1.

$$x \leftarrow \mathbf{H}(s\|\mathbf{H}(I\|" : "\|P))$$
$$v \leftarrow g^x \quad \bmod N$$

Figure B.1: *Deriving x and v as given in RFC 5054, where H represents a cryptographic hash function (for example, SHA256).*

Although it's infeasible to compute P from x or x from v, it's possible to use knowledge of v (and the public parameters) to test a candidate password, P'. All an opponent needs to do is compute v' from P' and see if v' equals v. If $v' = v$ then the guessed password P' is (almost certainly) the correct password.

As discussed elsewhere, we offer three defenses against such an attack if an attacker obtains v (which is stored on our server and transmitted during initial registrations).

- We use two-secret key derivation (2SKD) with the completely random Secret Key as one of the secrets in deriving x. Password cracking isn't a feasible approach for an attacker without the Secret Key. For a discussion of this point, see 3.2.

- We use a slower key derivation function for deriving x than the one shown in Figure B.1, so even if an attacker obtains both v and the user's Secret Key, each guess is computationally expensive.

- We encourage the use of strong account passwords. Thus an attacker who has both v and the account password will need to make a very large number of guesses.

The latter two mechanisms come into play only if the Secret Key is acquired from the user's device.

It should be noted that although the password processing shown in Figure B.1 is presented in RFC 5054[34], the standard does not insist on it. Indeed, RFC 5054 refers to RFC 2954[35] S3.1 which states

> SRP can be used with hash functions other than [SHA1]. If the hash function produces an output of a different length than [SHA1] (20 bytes), it may change the length of some of the messages in the protocol, but the fundamental operation will be unaffected...

> Any hash function used with SRP should produce an output of at least 16 bytes and have the property that small changes in the input cause significant nonlinear changes in the output. [SRP] covers these issues in more depth.

So in our usage, we compute x using the key derivation method described in detail in 8.2.

---

[34]Taylor et al. (2007)
[35]Rehbehn and Fowler (2000)

B.1 The math of SRP

B.1.1 Math background

The client will have its derived secret x, and the server will have its verifier, v. The mathematics that allow for the client and server to mutually authenticate and arrive at a key without exposing either secret is an extension of Diffie-Hellman key exchange (DHE). This key exchange protocol is, in turn, based on the discrete logarithm problem (DLP).

Recall (or relearn) from high school math:

$$(b^n)^m = b^{nm} \tag{B.1}$$

Equation (B.1) holds true even if we restrict ourselves to integers and do all of this exponentiation modulo some number N.

The crux of the discrete logarithm problem (DLP) is that if we pick N and g appropriately in equation (B.2)

$$v = g^x (\bmod N) \tag{B.2}$$

It's easy (for a computer) to calculate v when given x, but infeasible to compute x when given v.

Calculating x from v (given g and N) is computing the discrete logarithm of v. To ensure calculating the discrete logarithm is, indeed, infeasible, N must be chosen carefully. The particular values of N and g used in 1Password are drawn from the groups defined in RFC 3526.[36] Given current and anticipated computing power, N should be at least 2048 bits.

Figure B.2: *Sophie Germain (1776–1831) proved that Fermat's Last Theorem holds for exponents $n = 2q+1$ where both q and n are prime. Primes like q are now called 'Sophie Germain primes.' Germain stole the identity of a male mathematics dropout to enter into correspondence with mathematicians in France and elsewhere in Europe. It's only after they'd come to respect her work that she could reveal her true identity. Her fame at the time was mostly for her work in mathematical physics, but it her work in number theory that plays a role in cryptography today.*

B.1.2 Diffie-Hellman key exchange

If Alice and Bob have agreed on some g and N, neither of which need to be secret, Alice can pick a secret random number a and calculate $A = g^a (\bmod N)$. Bob can pick his own secret, b, and calculate $B = g^b (\bmod N)$. Alice can send A to Bob, and Bob can send B to Alice.

Assuming an appropriate N and g, Alice won't be able to determine Bob's secret exponent b, and Bob won't be able to determine Alice's secret exponent a. No one listening in – even with full

---

[36]Kivinen and Kojo (2003)

knowledge of g, N, A, and B – will be able to determine a or b.[37] There is, however, something that both Alice and Bob can calculate that no one else can. In what follows, it goes without saying (or writing) that all operations are performed modulo N.

Alice can compute:

$$S = B^a \tag{B.3}$$

$$= (g^b)^a \quad because \ \ B = g^b \tag{B.4}$$

$$= g^{ba} \quad because \ \ (B.1) \tag{B.5}$$

Equation (B.3) is what Alice actually computes because she knows her secret a and has been given Bob's public exponent. But note that the secret, S, that Alice computes is the same as what we see in (B.5).

Bob can compute:

$$S = A^b \tag{B.6}$$

$$= (g^a)^b \quad because \ \ A = g^a \tag{B.7}$$

$$= g^{ab} \quad because \ \ (B.1) \tag{B.8}$$

From equations (B.8) and (B.5) we see that both Alice and Bob are computing the same secret, S. They do so without revealing any secrets to each other or anyone listening in.

We use the session key, S, as an additional encryption and authentication layer on the client/server communication for that session. This is in addition to the encryption and authentication provided by TLS and the authenticated encryption of the user data.

All the secrets used and derived during Diffie-Hellman key exchange (DHE) are ephemeral: They're created for the individual session alone. Alice will create a new a for each session; Bob will create a new b for each session; the derived session key, S, will be unique to that session. One advantage of this is that a successful break of these secrets by some attacker will allow the attacker to decrypt the messages of that session only.

---

[37]There are numerous mathematical assumptions behind the claim that it's infeasible to determine a from A. Mathematicians are confident that some things involved are "hard" to compute but lack full mathematical proof. There are also some physical assumptions behind the security claims. We know the relevant computations we want to be difficult are not hard using large quantum computers of a certain sort. We're assuming, with some justification, that constructing the appropriate sort of quantum computer is beyond anyone's reach for at least a decade. We anticipate the development of post-quantum cryptographic algorithms over the next decade or so, but nothing is yet suitably mature to be of use to us now.
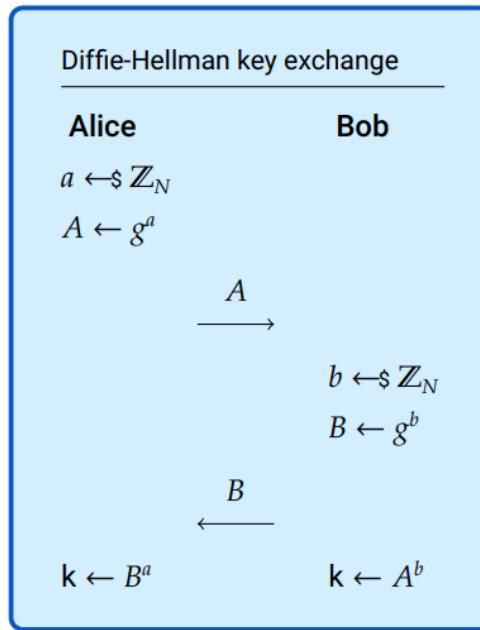
Figure B.3: *In Diffie-Hellman key exchange, N and g are already known to all parties and all exponentiation is done mod N. The form given here is somewhat less general than it could be in order to avoid having to introduce more notation and abstractions.*

B.1.3 Authenticated key exchange

Diffie-Hellman key exchange (DHE), as described in the previous section, allows Alice and Bob to agree on an encryption key for their communication. It doesn't, however, include a mechanism by which either Alice or Bob can prove to the other they are Alice and Bob. Our goal, however, is to have mutual authentication between the 1Password client and server.

In order for Alice to prove to Bob she's the same "Alice" he has corresponded with previously, she needs to hold (or regenerate) a long-term secret. At the same time, we don't want to transmit any secrets during authentication.

Secure Remote Password (SRP) builds upon Diffie-Hellman key exchange (DHE), but adds two long-term secrets. x is held (or regenerated) by the client and v, the verifier, is stored by the server. The verifier is created by the client from x and transmitted only during initial enrollment, and that's the only time a secret is transmitted.

As described in detail in "Key derivation," x is derived from a account password and Secret Key. The client computes $v = g^x$ and sends v to the server during initial enrollment.

During a normal sign-in, the client picks a secret random number a and computes $A = g^a$ as described above in "Diffie-Hellman key exchange". It sends A to the server (along with its email address).

The server picks a random number, b, but unlike unauthenticated Diffie-Hellman key exchange (DHE), it computes B as $B = kv + g^b$ and sends that to the client.

Everyone (including a possible attacker) can now compute a non-secret u from A and B by using a hash.[38] The server will calculate a raw S as

---

[38]It doesn't matter too much how u is created, but it must be standardized so the server and client do it the same way. We use the SHA256 hash of $A|B$.

$$S = (Av^u)^b \tag{B.9}$$

The client will calculate the same raw S as

$$S = (B - kg^x)^{a+ux} \tag{B.10}$$

The client and server will calculate the the same raw S if v is constructed from x as in equation (B.2) and A and B are constructed as described above. The proof is left as an exercise to the reader. (And the proof this is the only feasible way for the values to be the same is left for advanced texts.)

Appendix C

Appendix C: Verifying public keys

"Key verification is a weak point in public-key cryptography"[39]

At present, there's no robust method for a user to verify the public key they're encrypting data to belongs to their intended recipient.[40] As a consequence, it would be possible for a malicious or compromised 1Password server to provide dishonest public keys to the user and run a successful Man in the Middle (MITM) attack. Under such an attack, it would be possible for the 1Password server to acquire vault encryption keys with little ability for users to detect or prevent this.

Story 12 illustrates what might happen in the case of such an attack during vault sharing.

Story 12: Mr. Talk is the cat in the middle

Molly (a dog) joins a team, and as she does, generates a public key pair. Let's say the public key exponent 17 and public modulus 4171: $\mathsf{pk}_M = (17; 4171)$. (Of course in the actual system that modulus would be hundreds of digits long.) Only Molly has access to the corresponding private part, $d_M = 593$. When Patty (another dog) encrypts something using (17; 4171) only Molly, with her knowledge that d_M is 593 can decrypt it.

Now suppose Mr. Talk (the neighbor's cat) has taken control of the 1Password server and database. Mr. Talk creates another public key, $\mathsf{pk}_T = (17; 4183)$. Because Mr. Talk created that key, he knows the corresponding private part of the key, d_T, is 1905.

Patty wants to share a vault with Molly. Suppose that vault key is 1729. (In real life that key would be a much bigger number.) So she asks the server for Molly's public key. But Mr. Talk, now in control of the server, doesn't send her Molly's real public key — he sends the fake public key he created. Patty will encrypt the vault key, 1792, using the fake public key that Mr. Talk created. Encrypting 1729 with (17; 4183) yields 2016. Patty sends that up to the server for delivery to Molly.

Mr. Talk uses his knowledge of d_T to decrypt the message. So he learns the vault key is 1729. He then encrypts that with Molly's real public key, (17; 4147), and gets 2826. When Molly next signs in, she gets that encrypted vault key and is able to decrypt it using her own secret, d_M. The message she receives is correctly encrypted with her public key, so she has no reason to suspect anything went wrong.

Mr. Talk was able to learn the secrets Patty sent to Molly, but he wasn't able to learn the secret parts of their public keys.

---

[39] Free Software Foundation (1999)
[40] The role of public key encryption in 1Password is described in "How items are shared with anyone" and "Restoring a user's access to a vault."

> **Dangerous bend**
>
> The use of plain RSA (and small numbers) in Story 11 was to simplify the presentation. The underlying math of the RSA algorithm must be wrapped in a construction that addresses the numerous and large dangers of plain RSA.
>
> For those who wish to check the math of the story recall that:
>
> $d = e^{-1} \mod \lambda(N)$;[a]
>
> that for encrypting a message $c = e^m \mod N$;
>
> and that decrypting a ciphertext $m = c^d \mod N$.
>
> In our example $\lambda(4157) = \mathrm{lcm}(43 - 1, 97 - 1) = 672$,
>
> and $\lambda(4183) = \mathrm{lcm}(47 - 1, 89 - 1) = 2024$.
>
> ---
>
> [a] e is the public exponent and $\lambda(N)$ is the Carmichael totient, which can be calculated from p and q, the factors of N, as $\mathrm{lcm}(p - 1, q - 1)$.

For simplicity, Story 12 only works through adding someone to a vault, but the potential attack applies to any situation in which secrets are encrypted to another's public key. Thus, this applies during the final stages of recovery or when a vault is added to any group as well as when a vault is shared with an individual. This threat is probably most significant with respect to the automatic addition of vaults to the Recovery Group as described in "Restoring a user's access to a vault."

C.1   Types of defenses

The kind of problem we describe here is notoriously difficult to address, and it's fair to say there are no good solutions to it in general. There are, however, two categories of (poor) solution that go some way toward addressing it in other systems.

C.1.1   Trust hierarchy

The first defense requires everyone with a public key to prove the key really is theirs to a trusted third party. That trusted third party would then sign or certify the public key as belonging to who it says it belongs to. The user of the public key would check the certification before encrypting anything with that key.

Creating or using a trust hierarchy isn't particularly feasible within 1Password, as each individual user would need to prove to a third party their key is theirs. That third party cannot be AgileBits or the 1Password server – the goal is to defend against a Man in the Middle (MITM) attack launched from within the 1Password server. Although the 1Password clients could assist in some of the procedure, it would place costly burden on each user to prove their ownership of a public key and publish it.

C.1.2   User-to-user verification

The second approach is to enable users to verify keys themselves. They need to perform that verification over a communication channel that's not controlled by 1Password. Patty needs to talk directly to Molly, asking Molly to describe pk_{M_a} in a manner that will allow Patty to distinguish it from a maliciously crafted pk_{M_f}.

In the case of RSA keys, the crucial values may include a number that would be hundreds of digits long if written out in decimal notation. Thus a cryptographic hash of the crucial numbers is used, which is then made available presented in some form or other. Personal keysets also contain an Elliptic Curve Digital Signature Algorithm (ECDSA) key pair that's used for signing. These keys are far shorter than RSA keys, but may still be too large to be directly compared by humans.

Recent research[41] has confirmed the long suspected belief that the form of fingerprints makes comparisons difficult and subject to security sensitive errors. Such research does point to ways in which the form of fingerprints can be improved, and it's research we're closely following.

The difficulty for users with verifying keys via fingerprints isn't just the technicalities of the fingerprint itself, but in understanding what they're for and how to make use of them. As Vaziripour, J. Wu, O'Neill, et al. point out, "The common conclusion of [prior research] is that users are vulnerable to attacks and cannot locate or perform the authentication ceremony without sufficient instruction. This is largely due to users' incomplete mental model of threats and usability problems within secure messaging applications."[42]

Users may need to understand:

- Fingerprints aren't secret.

- Fingerprints should be verified before using the key to which they are bound.

- Fingerprints must be verified over an authentic and tamper-proof channel.

- That communication channel must be different from the communication system the user is trying to establish.

The developers of Signal, a well-respected secure messaging system, summarized some difficulties with fingerprints[43]

Publishing fingerprints requires users to have some rough conceptual knowledge of what a key is, its relationship to a fingerprint, and how that maps to the privacy of communication.

The practice of publishing fingerprints is based in part on the original idea that users would be able to manage those keys over a long period of time. This has not proved true, and has become even less true with the rise of mobile devices.

Although their remediation within Signal has a great deal of merit, only a small portion of Signal users attempt the process of key verification. When they're instructed to do so (in a laboratory setting) they often don't complete the process successfully.[44]

C.2 The problem remains

We're aware of the threats posed by Man in the Middle (MITM), and users should be aware of those, too. We'll continue to look for solutions, but we're unlikely to adopt an approach that places a significant additional burden on the user unless we can have some confidence in the efficacy of such a solution.

---

[41] Dechand et al. (2016)
[42] Vaziripour et al. (2018)
[43] Marlinspike (2016)
[44] Vaziripour et al. (2017) and Vaziripour et al. (2018)

Glossary

account password

Something you must remember and type when unlocking 1Password. It's never transmitted from your devices. Previously known as Master Password. 7, 8, 9, 10, 11, 14, 16, 30, 32, 33, 34, 35, 36, 37, 39, 44, 47, 48, 50, 51, 53, 55, 66, 69, 71, 73, 74, 77

Account Unlock Key (AUK)

Key used to decrypt a user's personal key set. It's derived from the user's account password and Secret Key. Previously known as the Master Unlock Key. 16, 30, 32, 33, 35, 36, 37, 39, 48, 50, 55, 56, 69, 70, 71, 73

Advanced Encryption Standard (AES)

Probably the best studied and most widely used symmetric block cipher. 15, 56, 60

authentication

The process of one entity proving its identity to another. Typically the authenticating party does this by proving to the verifier that it knows a particular secret that only the authenticator should know. 11, 12, 13, 14, 20, 25, 26, 27, 28, 30, 32, 33, 35, 37, 39, 40, 42, 51, 53, 54, 57, 60, 61, 62

BigNum

Some cryptographic algorithms involve arithmetic (particularly exponentiation) on numbers that are hundreds of digits long. These require the use of Big Number libraries in the software. 32

Chosen Ciphertext Attacks (CCA)

A class of attacks during which the attacker modifies encrypted traffic in specific ways and may learn plain text by observing how the decryption fails. 15

confidentiality

Data confidentiality involves keeping data secret. Typically this is achieved by encrypting the data. 11, 14, 61, 62

CPace

A modern PAKE using a shared secret, defined by Abdalla, Haase, and Hesse (CPace, a balanced composable PAKE). 40, 41

credential bundle

A bundle containing a randomly generated SRP-x and Account Unlock Key (AUK), used to sign in to 1Password when signing in with single sign-on (SSO). It's encrypted by the device key and stored on 1Password servers. See also Device Key 37, 39, 40, 42, 62

Cryptographically Secure Pseudo-Random Number Generator (CSPRNG)

A random number generator whose output is indistinguishable from truly random. Despite "pseudo" in the name, a CSPRNG is entirely appropriate for generating cryptographic keys. 29, 50

device key

A cryptographic key stored on a 1Password client that uses single sign-on (SSO). It's used to decrypt the credential bundle it receives from the server upon successful sign in. See SSO 37, 39, 40, 42, 72

Diffie-Hellman key exchange (DHE)

An application of the discrete logarithm problem (DLP) to provide a way for parties to decide upon a secret key without revealing any secrets during the communication. It's named after Whitfield Diffie and Martin Hellman who published it in 1976. 75, 76, 77

discrete logarithm problem (DLP)

If $y \equiv gx \bmod p$ (for a carefully chosen p and some other conditions) it's possible to perform exponentiation to compute y from the other variables, but it's thought to be infeasible to compute x from y. Computing x from y (and the other parameters) is reversing the exponentiation and is taking a logarithm. 75

Ellipic Curve Cryptography (ECC)

A public key encryption system able to work with much smaller keys than are used for other public key systems. 17

Elliptic Curve Digital Signature Algorithm (ECDSA)

The elliptic curve digital signature algorithm is digital signature algorithm based on elliptic curve cryptography described in FIPS PUB 186-4^8^. 80

Emergency Kit

Contains your Secret Key, account password, and details about your account. Your Emergency Kit should be printed and stored in a secure place, and used if you forget your account password or lose your Secret Key. 9, 10

end-to-end (E2E)

Data is only encrypted or decrypted locally on the users' devices with keys that only the end users possess. This protects the data confidentiality and integrity from compromises during transport or remote storage. 11, 14, 40, 66, 73

Galois Counter Mode (GCM)

An authenticated encryption mode for use with block ciphers. 15, 56, 60

hash-based key derivation function (HKDF)

A key derivation function that uses HMAC for key extraction and expansion. Unlike PBKDF2, it's not designed for password strengthening. 16, 23, 31, 32, 50, 51

HTTP Strict Transport Security (HSTS)

Strict Transport Security has the server instruct the client that insecure HTTP is never to be used when talking to the server. 61, 67

infeasible

Effectively impossible. It's not technically impossible for a single monkey placed in front of a manual typewriter for six weeks to produce the complete works of Shakespeare. It is, however, infeasible, meaning that the probability of it happening is so outrageously low that can be treated as impossible. 74, 75

integrity

Preventing or detecting tampering with the data. Typically done through authenticated encryption or message authentication. 11, 61

JSON Web Key (JWK)

A format for describing and storing cryptographic keys defined in RFC 7517. 16, 32

JSON Web Token (JWT)

A means of representing claims to be transferred between two parties and defined in RFC 7517. These are typically signed cryptographically. 57, 59

key encryption key (KEK)

An encryption key used for the sole purpose of encrypting another cryptographic key. 16

key set

How collections of keys and their metadata are organized within 1Password. 16, 17, 19, 28, 35, 36, 47, 49, 51, 52, 58, 69

linked app or browser

A client trusted to use SSO by having set up a device key and created a corresponding credential bundle. 40, 41

Man in the Middle (MITM)

A Man in the Middle attack has Alice believing she's encrypting data to Bob, while she's actually encrypting her data to a malicious actor who then re-encrypts the data to Bob. The typical defense for such an attack is for Alice and Bob to manually verify they're using the correct public keys for each other. The other approach is to rely on a trusted third party who independently verifies and signs Bob's public key. 68, 79, 80, 81

mutual authentication

Mutual authentication is a process in which all parties prove their identity to each other. 60, 61, 77

nonce

A non-secret value used in conjunction with an encryption key to ensure relationships between multiple plaintexts are not preserved in the encrypted data. Never encrypt different data with the same combination of key and nonce. Ideally, most software developers using encryption – as they should – would never have to interact with or much less understand the difference between them. We do not live in such a world. 56

passkey

A credential with which you authenticate to a server. Unlike a password, the passkey isn't sent to the server to authenticate. Instead, the passkey signs a challenge the server provides to your device. This process is also known as WebAuthn or FIDO2 authentication. 37, 39, 41, 42, 51, 62, 72

password-authenticated key exchange (PAKE)

Password-based key exchange protocol allows for a client and server to mutually authenticate each other and establish a key for their session. It relies on either a secret each have or related secrets that each have. 13, 50, 51, 73

primary account

A local client may distinguish a single account it knows about as the primary account. Unlocking this account may automatically unlock secondary accounts the client may handle. See also secondary account 69

Recovery Group

The 1Password Group that has a copy of the vault keys for vaults that may need to be recovered if account passwords or Secret Keys are lost. 47, 48, 49, 50, 68

Recovery Group member

A member of a Recovery Group. See Recovery Group 47, 48, 68, 72

RESTful

The adjectival form of representational state transfer (REST). See REST 53

salt

A non-secret value added to either an encryption process or hashing to ensure the result of the encryption is unique. Salts are typically random and unique. 16, 30, 31, 32, 33, 34, 35, 36, 73

secondary account

An account that a client may unlocked automatically when the primary account is unlocked. See also primary account 70

Secret Key

A randomly generated user secret key that is created upon first signup. It's created and stored locally. Along with the user's account password, it's required both for decrypting data and for authenticating to the server. The Secret Key prevents an attacker who has acquired remotely stored data from attempting to guess a user's account password. Previously known as the Account Key. 7, 8, 9, 10, 11, 14, 16, 30, 32, 33, 34, 35, 36, 37, 39, 44, 47, 48, 50, 51, 53, 56, 66, 67, 69, 71, 72, 73, 74, 77

Secure Remote Password (SRP)

A method for both a client and server to authenticate each other without either revealing any secrets. In the process, they also agree on an encryption key to be used for the current session. We're using Version 6 with a modified key derivation function. 11, 13, 32, 34, 37, 51, 60, 61, 62, 73, 74, 77

single sign-on (SSO)

In the setting of a company or another organization, when you are provided with a single set of username, password, or other authentication factors to log in to services that company or organization provides for you. It's one of the methods that can be used to sign in to 1Password. 37, 39, 40, 41, 42, 62, 72

slow hash

A cryptographic hash function designed to be computationally expensive. Used for hashing passwords or other guessible inputs to make guessing more expensive to an attacker who has the hash output. 32

SRP-v

The Secure Remote Password (SRP) verifier, v, used by the server to authenticate the client. 51

SRP-x

The client secret, x, used by the Secure Remote Password (SRP) protocol. Derived from the user's account password and Secret Key. 30, 33, 35, 36, 37, 39, 48, 51, 55, 56, 70

Transport Layer Security (TLS)

The successor to SSL. It puts the "S" in HTTPS. 13, 35, 60, 61, 62, 67

two-secret key derivation (2SKD)

Two different secrets, each with their own security properties, are used in deriving encryption and authentication keys. In 1Password, these are your account password (something you know) and your Secret Key (a high-entropy secret you have on your device). 7, 8, 14, 30, 47, 71, 73, 74

Unicode Normalization Form Compatibility Decomposition (NFKD)

A consistent normal form for Unicode characters that could otherwise be different sequences of bytes. 31

Universally Unique Identifier (UUID)

A large arbitrary identifier for an entity. No two entities in the universe should have the same UUID. 23, 25, 26, 33, 34, 35, 56

Changelog

v0.5.2 — 2026-01-15

- **New:**
    - Added Acknowledgments section.
    - Added PDF version of the current document.
    - Various formatting additions (for PDF appearance/layout, to improve glossary display).
- **Improved:**
    - Changed title of document to include "white paper."
    - Minor typographical corrections (e.g., uppercase letters to lowercase, missing and misplaced punctuation).

v0.5.1 — 2025-01-16

- **Improved:** Restore release, release date, and changelog.

v0.5 — 2024-12-14

- **New:** New primary format: `bookdown`
- **Improved:** Various typographical fixes and wording improvements throughout.

v0.4.7 — 2024-07-20

- **New:** Describe how the ability to regain access to an account with recovery codes works.

v0.4.6 — 2023-10-26

- **New:** Describe how unlocking 1Password with a passkey works.
- **Improved:**
    - Note the "Manage Groups" permission has equivalent cryptographic access as the Recovery Group.
    - Describe how 1Password 8 handles multiple accounts in an app in "Beware of the Leopard."
    - Add a caveat about replay protections in "Transport Security."
- **Corrected:** Improved two sentences in "How items are shared with anyone."

v0.4.5 — 2023-06-19

- **Improved:**
    - Version information is less gitty.
    - Corrected an erroneous sentence in "How items are shared with anyone."
- **Corrected:** Handshake values are now labelled correctly in the CPace diagram in the SSO chapter.

v0.4.4 — 2023-05-10

- **Improved:**
  - Improved line breaking.
  - Removed placeholder sections and chapters.

v0.4.3 — 2023-04-06

- **Improved:**
  - Updated name of item sharing.
  - More consistent naming.

v0.4.2 — 2023-03-31

- **New:** Describe how unlocking 1Password with SSO works.

v0.4.1 — 2023-02-17

- **New:** PBKDF2 iteration count increased from 100,000 to 650,000.
- **Improved:**
  - Removed empty placeholder chapter "How our servers are managed."
  - Use SHA256 in examples instead of SHA-1.
- **Corrected:**
  - Changelog date for 0.4.0 was off by a year.
  - Various typographical errors.

v0.4.0 — 2021-10-26

- **New:** New chapter: "How items are shared with anyone."
- **Improved:**
  - More consistent referencing of chapters and sections.
  - s/Master Unlock Key/Account Unlock Key/
  - Updated discussion of Secret Key storage in section Locally exposed Secret Keys.
  - s/Master Password/account password/
  - Hinted at OP8 resolution of unlocking one account with a different account password.
  - A few fewer typos.
- **Corrected:** Fixed the example calculation of d in and around Story 11 on 90.

v0.3.1 — 2021-04-19

- **Improved:**
  - Expanded 1Password Connect.
  - Fewer typos.

v0.3.0 — 2021-04-13

- **Improved:**

  - Cover page update.

  - Updated Transport Security, particularly with respect to pinning and EV certificates.

  - Updated Account password section in Account password and Secret Key to reflect current practice.

  - Title page and fonts should no longer break certain versions of Acrobat.

- **New:** New sections: 1Password Connect and Clients handling multiple accounts, on unlocking multiple accounts with one account password.

v0.2.10 — 2019-01-12

- **Improved:** No proprietary fonts (in preparation for opening source).

v0.2.9 — 2018-12-30

- **Improved:**

  - Fixed text citation in MitM appendix.

  - Less verbose bibliographic data in margins.

v0.2.8 — 2018-12-30

- **New:**

  - Appendix describing MitM threat.

  - Mention of MitM in "Beware of Leopard" appendix with pointer to new appendix.

- **Improved:**

  - Typos.

  - Margin citation format changes.

v0.2.7 — 2018-09-10

- **New:**

  - Mentioned ECDSA key creation.

  - Updated RNG table to include new clients, and switch from MSCAPI to CNG.

- **Improved:**

  - Further clarified web-client risks.

  - Corrected some errors and inconsistencies in SRP appendix.

  - Typos.

v0.2.6 — 2017-04-11

- **Changed:**

    - The Account Key has been renamed "Secret Key."

    - Substantially restructured presentation of authentication.

- **New:** New section: A modern approach to authentication should better communicate why 1Password's authentication is designed as it is and providing a top level view before getting into the details of key derivation.

- **Improved:**

    - Improved presentation of two-secret key derivation (2SKD)

    - Removed explicit comparison of 2SKD and multi-factor authentication (MFA).

    - s/Account Key/Secret Key/g, but not all graphics have been updated to reflect the new name.

    - Typos fixed and minor wording improvements throughout.

    - TeXnical changes to improve line breaking.

v0.2.5 — 2017-02-20

- **Improved:**

    - Even more typos fixed.

    - Even more minor wording and clarification improvements.

    - Document title reflects Teams, Families and individual accounts.

    - Text refers to "1Password" instead of "1Password for Teams and Families."

v0.2.4 — 2016-09-28

- **New:**

    - Detailed section on Restoring a User's Access to a Vault.

    - New presentation of two-secret key derivation (2SKD): AK and MP rewritten in terms of 2SKD.

- **Improved:**

    - Typography.

    - More detail in "Beware of Leopard."

    - More math in SRP.

    - Typos fixed and minor wording improvements throughout.

    - Many small improvements and expansions throughout the text.

- **Fixed:**

    - Many many typos and phrasing awkwardnesses.

    - Colors are no longer emetic.

v0.2.3 — 2015-11-27

- **New:**
  - Documented how server-side data is stored and which keys are exposed.
  - Expanded explanation of SRP authentication.
- **Fixed:** Numerous typos.

v0.2.2 — 2015-11-03

- **New:**
  - First public release
  - First release of the 1Password for Teams Security Design paper. This document will help you understand how we implemented all the awesome things in 1Password for Teams and how we keep your secrets, secret.

Bibliography

AgileBits. 2015. "OPVault Design." August 28, 2015. https://support.1password.com/opvault-design/.

Dechand, Sergej, Dominik Schürmann, Karoline Busse, Yasemin Acar, Sascha Fahl, and Matthew Smith. 2016. "An Empirical Study of Textual Key-Fingerprint Representations." In *25th USENIX Security Symposium (USENIX Security 16)*, 193–208. Austin, TX: USENIX Association. https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/dechand.

Free Software Foundation. 1999. "The GNU Privacy Handbook." The Free Software Foundation. 1999. https://www.gnupg.org/gph/en/manual.html.

Goldberg, Jeffrey. 2013. "You Have Secrets; We Don't: Why Our Data Format Is Public." AgileBits. May 6, 2013. https://blog.1password.com/you-have-secrets-we-dont-why-our-data-format-is-public/.

———. 2021. "How Strong Should Your Account Password Be? Here's What We Learned." September 27, 2021. https://blog.1password.com/cracking-challenge-update/.

Hunt, Troy. 2019. "Extended Validation Certificates Are (Really, Really) Dead." August 13, 2019. https://www.troyhunt.com/extended-validation-certificates-are-really-really-dead/.

Jackson, Collin, Daniel R. Simon, Desney S. Tan, and Adam Barth. 2007. "An Evaluation of Extended Validation and Picture-in-Picture Phishing Attacks." In *Financial Cryptography and Data Security*, edited by Sven Dietrich and Rachna Dhamija, 281–93. Berlin, Heidelberg: Springer Berlin Heidelberg.

Kivinen, T., and M. Kojo. 2003. "More Modular Exponential (MODP) Diffie-Hellman groups for Internet Key Exchange (IKE)." Request for Comments. Internet Engineering Task Force; RFC 3526 (Proposed Standard); IETF. http://www.ietf.org/rfc/rfc3526.txt.

Marlinspike, Moxie. 2016. "Safety Number Updates." Open Whisper Systems. November 17, 2016. https://signal.org/blog/safety-number-updates/.

Rehbehn, K., and D. Fowler. 2000. "Definitions of Managed Objects for Frame Relay Service." Request for Comments. Internet Engineering Task Force; RFC 2954 (Proposed Standard); IETF. http://www.ietf.org/rfc/rfc2954.txt.

Taylor, D., T. Wu, N. Mavrogiannopoulos, and T. Perrin. 2007. "Using the Secure Remote Password (SRP) Protocol for TLS Authentication." Request for Comments. Internet Engineering Task Force; RFC 5054 (Informational); IETF. http://www.ietf.org/rfc/rfc5054.txt.

Vaziripour, Elham, Justin Wu, Mark O'Neill, Ray Clinton, Jordan Whitehead, Scott Heidbrink, Kent Seamons, and Daniel Zappala. 2017. "Is That You, Alice? A Usability Study of the Authentication Ceremony of Secure Messaging Applications." In *Symposium on Usable Privacy and Security (SOUPS)*.

Vaziripour, Elham, Justin Wu, Mark O'Neill, Daniel Metro, Josh Cockrell, Timothy Moffett, Jordan Whitehead, Nick Bonner, Kent Seamons, and Daniel Zappala. 2018. "Action Needed! Helping Users Find and Complete the Authentication Ceremony in Signal." In *Fourteenth Symposium on Usable Privacy and Security (SOUPS 2018)*, 47–62. Baltimore, MD: USENIX Association. https://www.usenix.org/conference/soups2018/presentation/vaziripour.

Acknowledgments

This document was created using bookdown, an R package for authoring books and technical documents.

The interactive glossary functionality within the online version is powered by the glossary R package created by Lisa DeBruine.

Document Information

- **Format:** Created with R Markdown and bookdown
- **Source:** Available on GitHub and 1pw.ca
- **License:** Content © AgileBits d.b.a. 1Password Inc.