1Password Security Design

Teams, Families, and Individual Accounts

Key Security Features

1Password offers a number of notable security features, including

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

Server ignorance We are never in the position of learning your Master Password or your cryptographic keys. Details are in A modern approach to authentication.

Nothing "crackable" is stored Often a server will store the password hash. If captured, this can be used in password cracking attempts. Our **two-secret key derivation** mixes your locally held Secret Key with your Master Password so that 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 is 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 do not have those keys, so sharing decisions come from you. See How Vault Items Are Securely Shared for details.

Team managed data recovery We do not have the ability to recover your data if you forget your Master 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.

Contents

List of Stories

List of Asides

List of Tables

List of Figures

Principles

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

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

> **Privacy by Design** *It is 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.*

PRINCIPLE 1 Privacy by Design

You will find Principle 1 exemplified throughout our system, from our inability to acquire your Master Password during authentication (through use of Secure Remote Password (SRP) to our use of two-secret key derivation (2SKD) which means that we aren't in a position even attempt to crack your Master Password. Likewise, our use of end-to-end encryption protects you and your data from us or someone who gains access to our servers.

Our Principle 2 follows directly from the first:

> **Trust the Math** *Mathematics is more trustworthy than people or software. Therefore, when designing a system, prefer security that is enforced by cryptography instead of enforced by software or personnel policy.*

PRINCIPLE 2 Trust the Math

It is cryptography that prevents one person from seeing the items that they are 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 that 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 themselves security experts. This is more than just building a product and system that people will want to use, it is part of the security design itself:

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 is 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 will read this document, but everyone is entitled to security whether or not they seek to understand how it works.

The underlying mechanisms for even apparently 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, and instead 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.

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

A security system should be subject to public scrutiny. If the security of a system were to depend on some aspects of the design being kept secret, then 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.

[1] Goldberg, *You have secrets; we don't: Why our data format is public*.

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 on Beware of the Leopard.

This document will on occasion go into considerable technical detail that may not be of interest to every reader. And so, following the conventions developed in *The TₑX Book*,[2] some sections will be marked as a dangerous bend. Most of the text should flow reasonably well if you choose to ignore such sections.

[2] Knuth, *The TₑXbook*.

Many of those dangerous bend technical discussions involve explaining the rationale for some of the very specific choices we made in 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 is possible to pick the wrong tool for the specific job or to use it incorrectly. We feel that it is important not just to follow the advice of professional cryptographers, but to have an understanding of where that advice comes from. That is, it is important to know your tools (Principle 5).

Know your tools   *We must understand what security properties a tool does and doesn't have so that we use the correct tool the right way for the particular task.*

PRINCIPLE 5 Know your tools

Throughout this document there will also 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.

Master Password and Secret Key

1Password is designed to help you and your team manage your secrets. But there are some secrets that you need to take care of yourself in order to be able to access and decrypt the data managed by 1Password. These are your Master Password and your Secret Key[3], which are introduced in this section.

Decrypting your data requires *all* three of the following: your Master 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 that all three are needed, your data is protected by a combination of the best of each. Your Master Password and your Secret Key are your two secrets used in a process we are calling "two-secret key derivation."
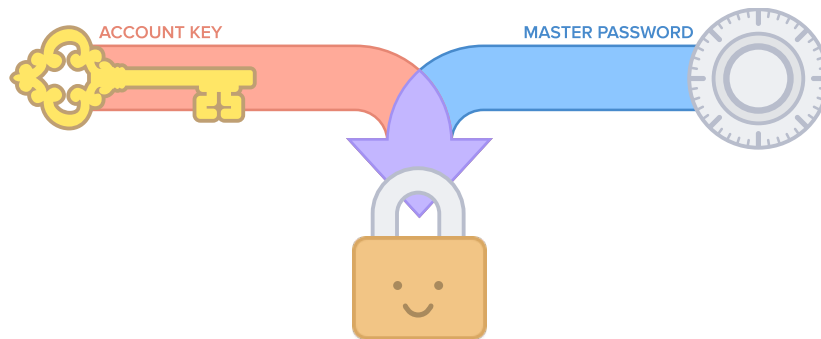
Figure 1: Two-secret key derivation combines multiple secrets when deriving authentication keys and encryption keys.

Master Password

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

The great limitation of secrets that must be remembered and used by actual humans is that they tend to be guessable by automated pass-
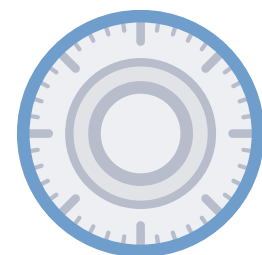
Figure 2: Your Master Password, like a combination to a lock, is something that you know.

word guessing systems. We do take substantial steps to make things harder for those attempting to guess passwords (see Slow hashing in Strengthening User Master Passwords), but it is impossible to know the capabilities that a well-resourced attacker may be able to bring to bear on password cracking.

1Password will suggest a Master Password for you when you create your account. In its current form the suggested passwords are words drawn uniformly and randomly from a list of more than 18000 English words. A Master Password with four such words will be one of 18000^4 (about 100 million billion) possibilities,[4] providing about 56 bits of entropy. The suggested password is generated entirely on your device using a cryptographically appropriate random number generator.

We very strongly encourage people to practice their Master Passwords and to write down a copy to be stored in a secure place.

Although we take substantial measures to make it difficult for password guessing systems to guess your Master Password, passwords that humans can remember and use are guessable. Even our automatically generated passwords may still be within reach of a highly resourced attacker if they should get hold of your encrypted data. This is why 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 in defending you in case of a server breach.

[4]Names for large numbers will follow the US short scale conventions, in which "one billion" means one thousand million. 100 million billion is 10^{17}.

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 will 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 is encrypted with a randomly chosen 256-bit AES key. There is no way he will ever be able to guess that. But the private key is encrypted with a key derived from your Master Password (and other stuff) so he figures that if he can guess your Master 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 Master 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 in the same way that an incorrect guess will fail without the Secret Key.

Oscar has discovered — much to his chagrin and our delight — that even all the data held by AgileBits is insufficient to verify a correct guess at someone's Master 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 is useless to him.

If Oscar had read this document he would have learned that he cannot learn or guess your Master Password or Secret Key from data held or sent to AgileBits.

Secret Key

Your Master Password is one of the secrets used in 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-798JRY-LJVD4-23DC2-86TVM-H43EB. This is uncrackable, but unlike your Master Password isn't something that people could be expected to memorize or even to type on a keyboard regularly.

Figure 3: Your Secret Key is a high-entropy secret that you have on your devices

The hyphens are not part of the actual Secret Key but are 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 will 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 Keys[5], all equally likely, thus placing them well outside the range of any sort of guessing. But while the Secret Key is unguessable it is 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.

[5]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.

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 Master 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. What this means for you is that you must not forget or lose the secrets that you need to access and decrypt your data. This is why 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.

Your Emergency Kit is a piece of paper (once you've printed it out) that will contain your account details, including your Secret Key. Figure 4 shows an example. It also has space for you to write your Master Password. If you are uncomfortable keeping both your Secret Key and Master Password on the same piece of paper, you may decide to store a written back up of your Master Password separately from your Emergency Kit.

A challenge we face is in finding ways to encourage people to actually
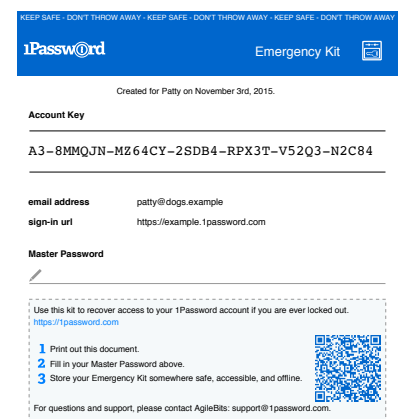
Figure 4: Sample Emergency Kit

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

It's been lonely in this safety deposit box all of 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 that 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 on "Save Emergency Kit". She probably thought that she would never need me, but she dutifully (and wisely) printed me out and wrote her Master 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 am the genie released from long imprisonment. Today I will 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 Master 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 of the account details, including the Secret Key. All that Alice will have to do is type in her Master Password.

What a day! Now I am being returned to the bank vault. I hope that Alice will not have reason to call upon me again, but we both feel safe knowing that I am here for her.

print and save their emergency kits. During the 1Password beta period we've added a number of places in which we nudge people toward this. This includes making it clearer what should be done with the Emergency Kit as in Figure 5, and by incorporating it among a set of "quests" users are to encouraged to complete after first starting to use 1Password.

1 Print it out

2 Fill it in

3 Keep it safe

Figure 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.

A modern approach to authentication

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

None-the-less, there is an authentication component to 1Password. For the most part its job is to ensure that an indvidual only receives the encrypted data that 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 is manifest in the the design of 1Password in three ways:

1. Our overall design is fundamentally based on E2E encryption.

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

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

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

What we want from authentication

Your Master Password and Secret Key are used to derive a key that is used for authenticating with the service we host.[6] Our service needs to know that you are who you say you are. Your 1Password client needs to

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

prove to the service that it has your Master Password and Secret Key. Only then will it give you access to your encrypted data (which your client still must decrypt) or to other administrative rights.

When Alice authenticates to Bob, she does so by proving that she has access to a secret[7] that only she should have access to. Bob performs some analysis or computation on the proof that Alice offers to verify it does demonstrate Alice's access to her secret.

We want an authentication system to allow Alice to reliably authenticate herself to Bob without introducing other security problems, and so there are a number of security properties we would like from a good authentication system. These are listed in Figure 7.

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

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 is correct according to its records. If the server determines that it is correct, it will consider that user authenticated.

The simplest way to prove you know a secret is to say it, and that is 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 figure 6.

This traditional design has a number of shortcomings. Indeed, it only satisfies the first authentication disideratum in figure 7. 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 Figure 6 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 the exchange in Figure 6 that would correspond to Bob not really being the castle guard, and Alice revealing to her password to an enemy.

In a typical Internet log in 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 do not wish to rely on it.

[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 identify*]
ALICE: My password is "xyzzy." [*She provides proof.*]

[BOB *verifies that that is the correct password for* ALICE.]

BOB: You may enter. [BOB *raises the portcullis and* ALICE *enters.*]

Figure 6: A very traditional authentication exchange

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 the server to

prove to each other that they each posses the appropriate secret without either of them revealing any secrets in the process. This is done by using a password-authenticated key exchange (PAKE).

Figure 7: Desired security properties of an authentication system. There is substantial overlap among these, but they are technically separate.

> ### Sign-in system desiderata
>
> *Prove client ID* Proves to the server that the user holds the user's secret.
>
> *Prove server ID* Proves to the user that the server holds the server's secret.
>
> *Eavesdroper safe* Does not reveal any information about either secret to anyone listening in on the process.
>
> *Not replayable* Cannot be replayed by someone who has recorded the process and wishes to repeat the exchange to fake a sign-in at some later time.
>
> *No secrets received* Does not 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 attack.

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. Furthermore, the puzzles are created jointly and uniquely during each session so that it is 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 "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[8] of encryption in our communication between client and server.

A well-designed PAKE – we use SRP, detailed in Secure Remote Password – can satisfy all of the requirements we've listed except for one. On its own, a PAKE would still leave something crackable on the server, something which would not be acceptable to us.

[8]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.

Making verifiers uncrackable with 2SKD

A 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 which is mathematically related to a long term authentication secret used by the client. Although this verifier is not 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), then the verifier stored by the server could be used to help test guesses of that user 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 do not feel that those steps are sufficient on their own.

   Our solution to ensure that data held by the server is not sufficient to launch a password cracking attempt on a user's Master Password is our use of two-secret key derivation (2SKD). An attacker who captures server data would need not only to make guesses at a user's Master Password but would need to have the user's 128-bit strong Secret Key.

   It is for this final desideratum that we have introduced 2SKD. With this, the information held on our systems cannot be used to check whether a Master Password guess is correct or not. Table 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	✗	✗	✗	✓

Table 1: Authentication schemes and what they do for you. The "+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 two-secret key derivation, as done in 1Password. The first column lists the desired security properties from Figure 7.

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

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 will be 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 (CCAS).

The vault key is used to encrypt each item in the vault.

Items contain overviews and details which are encrypted separately by the vault key.[9] We encrypt these separate so that we can quickly decrypt the information needed to list, sort, and find items without having to first decrypt everything in the vault.

Item overviews include the item fields needed to list items and to quickly match items to websites, such as Title, URLs, password strength indicator, and tags. Information that is 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 do not need to be used to list or quickly identify them, such as passwords and contents of notes. Documents are dealt with separately and will be discussed in Securing Documents.

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 a key encryption key (KEK) that is derived from your Master 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.

NewVault(items)

$K_v \leftarrow_\$ \text{KGen}()$

/ Add items to V

$V \leftarrow \varnothing$

for $x \in$ items

$\quad y_i \leftarrow \text{Enc}(K_v, x)$

$\quad V \leftarrow V \cup \{y_i\}$

endfor

return V

Figure 8: Algorithm for creating and populating a vault

[9]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).

Key derivation overview

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

Your Master Password will be trimmed and normalized. A non-secret salt is combined with your email address[10] and other non-secret data using Hash-based Key Derivation Function (HKDF) to create a new 32-byte salt.

Your Master Password and the salt are passed to PBKDF2-HMAC-SHA256 with 100,000 iterations. This will result in 32 bytes of data, which will be 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 Master Password.

The resulting 32 bytes of material (derived from both your Master Password and your Secret Key) will be your Master Unlock Key (MUK) which is used to encrypt the key (your private key) that is 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.

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.

[10]The reasons for binding your encryption key tightly with your email address is discussed in Restoring a User's Access to a Vault.

A first look at Key Set

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

```
"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
```

A common type of Key Set will have the structure listed in figure 9. When we speak of encrypting a Key Set we generally mean encrypting the symmetric key that is used to encrypt the private key.

Key Sets 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,

Figure 9: 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.

and public key encryption is RSA-OAEP with 2048-bit moduli and a public exponent of 65537.

Story 3: 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 that only she could use it; and 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 Master 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.

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 Sets 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 of the clients, we can manage upgrades without enormous difficulty.

Story 4: 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 The next day we ensure that all of 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 also limiting the scope of downgrade attacks isn't easy. But as illustrated in story 4, we are the single responsible entity for issuing clients and managing the server, and so through a combination of client and server policy we can defend against downgrade attacks.

How Vault Items Are Securely Shared

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 item with the public key of the recipient.

Story 5: 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 for Teams 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 willl 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.

1Password for Teams server never has a copy of the decrypted vault key and so is never in a position to share it. Only someone who has that key can encrypt a copy of it. Thus, an attack on our server could not result in unwanted sharing.

Getting the message (to the right people)

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

1Password for Teams 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 is done

without giving us the ability to acquire user secrets is the subject of the
next section.

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 are easily guessed. In this section we describe not only the details of the encryption algorithms and protocols used; but more importantly we describe 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 Vault Items Are Securely Shared. It is one thing to assert (as we have) that our design means that we never learn any of your secrets, but it is another to show how this is the case. As a consequence, this section will be substantially more technical than the others.

This section is the most technical of this entire document. You have been warned.

Key creation

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

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

Table 2: 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.

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.

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 that 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}.

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

Key derivation

For expository purposes it is 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.

Deriving two keys from two secrets

As discussed in Master Password and Secret Key, 1Password uses two-secret key derivation (2SKD) so that data stored by us cannot be used in brute force cracking attempts against the user's Master Password. The two secrets held by the user are their Master Password and their Secret Key.

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

The processes for deriving each of these are similar, but they involve different salts in the key derivation function. A user will have a salt that is used for deriving the MUK and a different salt that is used for deriving the SRP-x.

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

Preprocessing the Master Password

Before the user's Master 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.

Master Passwords are first stripped of any leading or trailing whitespace. (Step 3 Figure 10.) Whitespace is allowed within the Master

Key Derivation

1 : $p \leftarrow$ Master 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 : $k_m \leftarrow \textbf{PBKDF2}(p, s, 100000)$

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 10: The MUK is derived from the Master Password, p; Secret Key (previously known as Account Key), $k_{A'}$; and several not secrets including the account ID, I; and a salt, s.

Password, but because leading or trailing whitespace may not be visible to the user, we wish to avoid them creating a Master Password with such a space that they are unaware of.

Then it is normalized (Step 4) to a UTF-8 byte string using Unicode Normalization Form Compatibility Decomposition (NFKD) normalization. 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 1.

Discussion 1: Non-ASCII passwords

People quite naturally would like to be able to use passwords that involve characters other than the 7-bit US-ASCII printable characters. Yet doing so poses difficulties that simply supporting Unicode does not 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 is 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. Ex-

actly how it is 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 will be passed to it, 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, we will treat them as identical. In the case of "Å", the normalization that we have chosen, NFKD, will convert any of those three representations to 0x0041030A.

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 must exercise care in the construction of Master Passwords that go beyond the unambiguous 7-bit US-ASCII character set.

Preparing the salt

The 16-byte salt is then stretched using Hash-based Key Derivation Function (HKDF) and is itself 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.

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

Slow hashing

The normalized Master Password is then processed with PBKDF2-HMAC-SHA256, along with a salt.

⬙ The choice of PBKDF2-HMAC-SHA256 as our slow hash is largely a function of there being (reasonably) efficient implementations available for all of 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 all of our key derivation is performed by the client (so that the server never needs to see the password) we are constrained in our choices by our least efficient client. The Makwa password hashing scheme,[12] however, is a possible road forward because it allows some of the computation to be passed to a server without revealing any secrets to that server.

[12]Pornin, *The MAKWA Password Hashing Function*.

In the current version, there are 100,000 iterations of PBKDF2. Some estimates of cracking times are provided in the appendix on Strengthening User Master Passwords.

Combining with the Secret Key

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 Master Password. This is done using HKDF, using the raw Secret Key as its entropy source, the account ID as its salt, and the format version as additional data.

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

Figure 11: The MUK is represented as a JSON Web Key (JWK) object. It is given the distinguished key ID of "mp".

The resulting bytes from the use of HKDF will then be XORed with the result of the PBKDF2 operations. This will then be set with the structure of a JWK object as illustrated in figure 11.

Deriving the authentication key

The process of deriving the client-side authentication secret that is used for authenticating with the 1Password for Teams server is nearly identical to the procedure described above for deriving the MUK. The only difference is that an entirely independent salt is used for the PBKDF2 rounds. This ensures that 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). In the browser we use the JSBN library, and for all other platforms, we use the tools from OpenSSL.

THE ASTUTE READER may have noticed that the defender needs to perform 200,000 PBKDF2 rounds while an attacker (who has managed to obtain the Secret Key) "only" needs to perform 100,000 PBKDF2 rounds per guess, thus giving the attacker a 1-bit advantage over the defender in such an attack.

However, the sequence described above, in which the defender needs to derive both keys, is rarely performed. In most instances, the SRP-*x* will be encrypted with the MUK (or by some other key that is encrypted with the MUK) and stored locally. Thus the defender needs to derive only the MUK. Indeed, the client needs to go through both derivations only at original sign-up or when enrolling a new client.

Initial sign-up

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

When the invitation is created, the server will generate an account ID and will know which team someone has been invited to join and the type of account that is 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 is known to the team administrator. An invitation token is created by the server and is not made available to the administrator. Other information about the status of the invitation is stored on the server.

The user will be given (typically by email) the invitation UUID along with the invitation token and will use those to request from the server the invitation details. 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, the invited email address, and (if supplied by the inviter) the real name of the user. If it does not find a valid and active invitation for that UUID, it will return an

error.

THE CLIENT WILL GATHER and compute a great deal of information, some of which will be sent to the server.

1. Generate Secret Key ξ 🔑

2. Compute MUK

   (a) Generate encryption key salt ξ 🔳
   (b) Derive MUK from encryption salt, Master Password and Secret Key as described in Key derivation. 🔑

3. Create encrypted account key set

   (a) Generate private key ξ 🔑
   (b) Compute public key (from private key) 🔑 🔳
   (c) Encrypt private part with MUK 🔒 🔳
   (d) Generate key set UUID ξ 🔳
   (e) Include key set format 🔳

4. User information 🔳

   (a) Given name 🔳
   (b) Family name 🔳
   (c) Image avatar 🔳
   (d) Email 🔳

5. Device information 🔳

   (a) Generate device UUID ξ 🔳
   (b) Operating system (if available) 🔳
   (c) User agent (if applicable) 🔳
   (d) Hostname (if available) 🔳

6. Construct SRP verifier

   (a) Generate authentication salt ξ 🔳
   (b) Derive SRP-x from Master Password, Secret Key, and authentication salt 🔑
   (c) Compute SRP verifier from SRP-x. 🔑 🔳

7. Send to the server everything listed as '🔳'.

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

Table 3: Symbols used to indicate status of different data client creates during signup

Protecting email invitations

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

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 her Master 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 her 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 12

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

Figure 12: An add link contains the email address, the team domain, and the Secret Key

This new client will not have its salts nor its key derivation parameters and so must request those from the server. It will be able to generate its device information and create a device UUID.

```
{ "accountKeyFormat" : "A3",
  "accountKeyUuid" : "GWM4R8",
  "sessionID" : "TRYYDRPO2FDWRITHY7BETQZPN4",
  "status" : "ok",
  "userAuth" : {
      "alg" : "PBES2g-HS256",
      "iterations" : 100000,
      "method" : "SRPg-4096",
      "salt" : "WSwigQtQpxqYAri592W1lg"
  } }
```

Figure 13: Example response from server to auth request. Note that internally the Secret Key is often referred to as "Account Key."

The client will initiate an auth request to the server, sending the email address and the device UUID. A typical server response will look something like what is shown in Figure 13.

Once the client has the salt used for deriving its authentication secret, it can compute its SRP-x from that salt, the Master Password, and the Secret Key. During authentication, neither the client nor the server reveals any secrets to the other; and once authentication is complete, our own transport layer encryption is invoked on top of what is provided by Transport Layer Security (TLS). However, in the discussion here, we will 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 can request its encrypted personal key set from the server.

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

Figure 14: Overview of personal key set. The value of `encryptedBy` here indicates that the encrypted symmetric key is encrypted with the Master Unlock Key (MUK).

If the client has successfully authenticated, the server will allow it to fetch the key sets associated with the account. The personal key set will have the overall structure shown in Figure 14.

This contains an encrypted private key, the associated public key, and an encrypted symmetric key that is used to encrypt the private key. The encrypted symmetric key is encrypted with the MUK, using the parameters and the salt that are included with the encrypted symmetric key as shown in Figure 15.

```
{ "encPriKey" : { "..." },
  "encSymKey": {
      "kid": "mp",
      "enc": "A256GCM", "cty": "b5+jwk+json",
      "iv": "X3uQ83tlrdNIT_MG",
      "data": "gd9fzh8lqq5BYdGZpypXvMzIfkS ...",
      "alg": "PBES2g-HS256", "p2c": 100000,
      "p2s": "5UMfnZ23QaNVpyeKEusdwg" },
  "encryptedBy" : "mp",
  "pubKey" { "..." }, "uuid" : "c4pxet7a..." }
```

Figure 15: The encrypted symmetric key is encrypted with the Key used to encrypt a user's private key. It is derived from the user's Master Password and Secret Key, 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 AES-256-GCM.

The details of the public and private keys are illustrated in Figure 16.

```
{ "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 16: The public/priviate parts are specified using JWK.

Normal unlock and sign-in

When unlocking and signing in to 1Password for Teams from a client that has previously signed in, the client may[13] have everything it needs locally to compute its master unlock key and to compute or decrypt SRP-*x*. The client may already have the salts, encryption parameters, and its encrypted personal key set.

[13] 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.

Once the user enters a correct Master Password and the client reads the Secret Key, it can compute the MUK, decrypt the user's private key, and then decrypt any locally cached data. Depending on the completeness of the cached data, the client may be able to function offline.

Revoking Access

> **Incomplete Content**
>
> We're sorry. This section of this document is not yet ready. Anything you see in this section is at most an outline of things to come.

When Alice tells Bob a secret and later regrets doing so, she cannot make Bob forget that secret without resorting to brain surgery. We feel that brain surgery is beyond the scope of 1Password,[14] and therefore users should be aware that once a secret has been shared the recipient cannot be forced to forget that secret.

[14]We have made no formal decision on whether rocket science is also beyond its scope.

> **Story 6: 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 be getting access to a new team and a 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 that 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, "More cow bell", for the vault. "More cow bell" will be encrypted with a key that is encrypted by the vault key, but Tom should never get a copy of the encrypted Document.
>
> 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 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.

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 will use that language here; however it should be noted that the mechanisms by which these controls are enforced are not 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.

Cryptographically enforced controls

If someone has not been given access to a vault, it is impossible in all practical terms for them to decrypt its data. So at the simple level, if a user has not been added to a vault, the mathematics of cryptography ensure that they will not be able to decrypt it.

Because the server never has access to decrypted vault keys, it cannot 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.

Which controls are cryptographically enforced

Among mechanisms that are cryptographically enforced are

- Unlocking a vault

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

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

- Server does not learn user's Secret Key or Master Password

> **Incomplete Content**
>
> We're sorry. This section of this document is not yet ready. Anything you see in this section is at most an outline of things to come.

Server-enforced controls

A user (or their client) who has access to the vault key is not prevented by the cryptography from adding, deleting, or modifying items in that vault when the information resides locally on their device. The same key that they use to decrypt the data could be used to encrypt modified data.

However, one feature that 1Password offers is 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 7.

Story 7: A day in the life of read-only data

Patty (a clever but sneaky dog) has been granted access to a vault called "Locations" containing the locations of the water dish and the dog door. So has another member of the team, Morgan le Chien.

Patty thinks that maybe she will have the place to herself if Morgan can't manage to settle in. So she would 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 that Morgan would get that information instead.

But then 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. And so 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 so can manage remarkably well even if she doesn't know the location of the water bowl.

Which controls are enforced by server policy

> **Incomplete Content**
>
> We're sorry. This section of this document is not yet ready. Anything you see in this section is at most an outline of things to come.

Client-enforced policy

Client-enforced policies are those limitations which are 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 of our control, these policies may be circumvented by a malicious client or a determined user. This doesn't reduce their usefulness to ordinary users and may help prevent unintended disclosures or accidental actions.

See Story 8 for an illustration of what client-enforced policies can and cannot do.

Which controls are enforced by client policy

Each of the client policies requires that 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 `Write` permission is enforced by the server, not by 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 which are not 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 and create files which are not controlled by 1Password and printing out those files. This permission may be used to prevent accidentally disclosing the contents of an entire vault.

Multiple layers of enforcement

Something which is enforced by cryptography may also be enforced by the server, and something that is enforced by the server may also be enforced by the client. For example, the server will not 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 that the server would refuse to supply. Throughout this document we will typically mention only the deepest layer of enforcement without also listing the further ones.

Story 8: A day in the life of a concealed password

The administrators have come to be wary of how the dog Patty (see Story 7 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 do not want Patty to give that password to any of her friends should her paws accidentally press the "reveal" button.

And 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 cannot accidentally press 1Password's "reveal" button while her friends are watching. This is protected by client policy.

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

The house is overrun with Patty's friends running wild, and the administrators have learned an important lesson that client policy controls are easily evaded.

Administrator Roles and Powers

See Restoring a User's Access to a Vault for a discussion of how Team and Family administrators can perform account recovery for those who lose or forget their Master Password or Secret Key.

Incomplete Content

We're sorry. This section of this document is not yet ready. Anything you see in this section is at most an outline of things to come.

Restoring a User's Access to a Vault

If Albert forgets his Master Password or loses his Secret Key, it is impossible to decrypt the contents of his vaults unless those vaults have been shared with someone else who has not forgotten nor lost her Master Password and Secret Key. Our use of two-secret key derivation (2SKD) increases the risk to data availability as in addition to the possibility of a user forgetting her Master Password there is also the possibility that the Secret Key gets lost. Data loss can be catastrophic to a team, so some recovery mechanism is needed.

Our security design also requires that we, AgileBits, never have the ability to decrypt your data, and so we do not have the ability to restore anyone else's ability to decrypt his data if he has forgotten his Master Password or lost his Secret Key. Our solution is to place the power to recover access to vaults where it belongs: within the team.

Overview of Groups

To understand how the Recovery Group works it is first necessary to understand how a group works. A group will have a key set that is similar in nature to an individual's key set. It is 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.

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[15], *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. However, if *A* has not been granted access to a vault, she will be prevented by server policy from obtaining the vault data even though she has the key to that vault. Simple.

[15]For some values of the word "simply".

Recovery Groups

One of the most powerful capabilities that 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 Recovery Group. In 1Password Families there is no ability to separate the roles of Owner, Administrator, and Recovery Group member; they are all wrapped up as "Organizer." With 1Password Teams Administrators are given more control, but not all of 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.

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 are able to 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 of the vaults.

Recovery Group members never have the ability to learn anyone's Master Password, Secret Key, Master Unlock Key (мuк), or SRP-*x*. Recovery is recovery of the vault keys; it is not recovery of Master Passwords nor Secret Keys.

Protecting vaults from Recovery Group members

Although there is a chain of keys and data that would allow any member of the Recovery Group to decrypt the content of any vault, there are a number of mechanisms that prevent that:

1. A member of the Recovery Group will not be granted access to the encrypted data in a vault that they otherwise wouldn't have access to even if they can obtain the vault key.

2. 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 who is 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 is able to trick the server into delivering the encrypted vault keys when it shouldn't, the attacker would still need to obtain the vault data encrypted with that key.

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 that they use secure and free of malware.

- Members of the Recovery Group should be aware of social engineering trickery.

- Requests for recovery 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.

Figure 17 provides an overview of what data and keys are held by whom. In order to keep the diagram manageable, some hopefully irrelevant details have been omitted. For example the transport encryption layers for the messages is entirely skipped (see Transport Security) and where we speak of encrypting or decrypting private keys, it is actually encrypting or decrypting the keys that the private parts of keys are encrypted with (see Figure 9). 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 keyset, (pk_B, 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
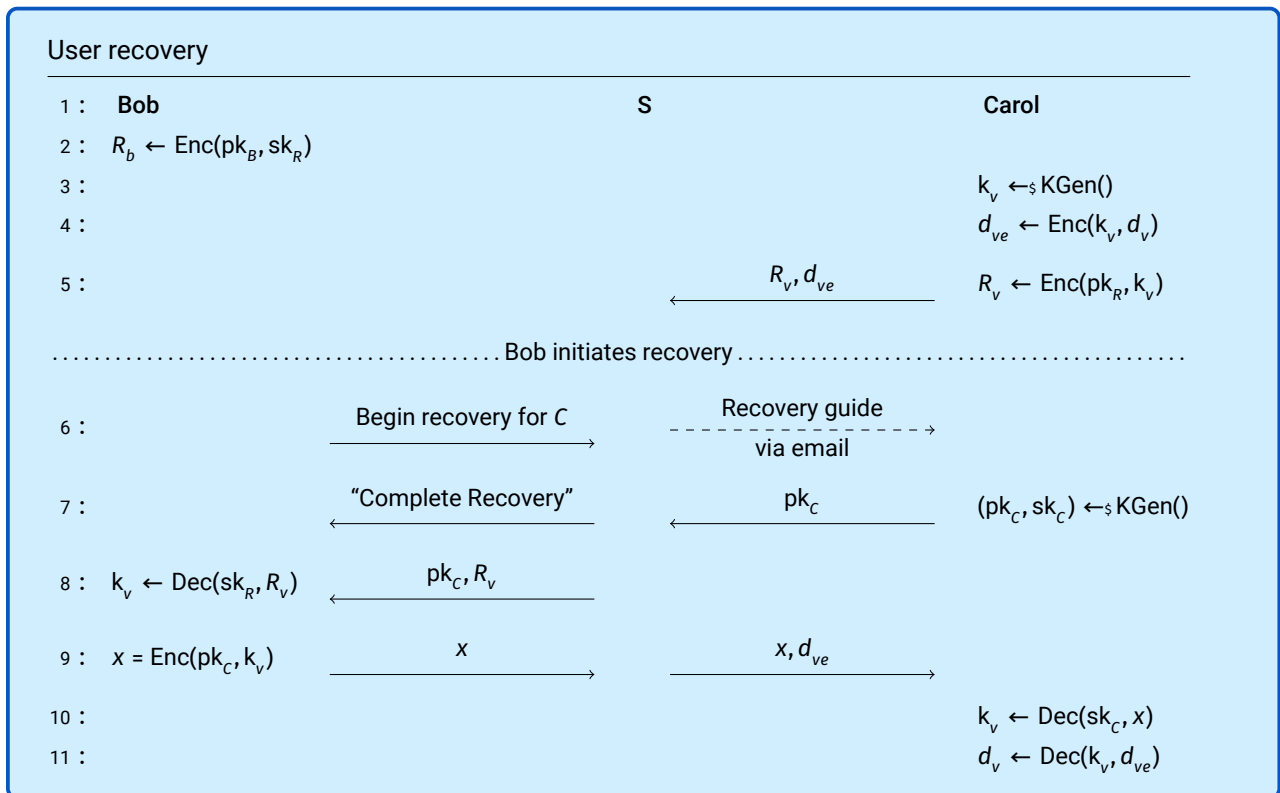
Figure 17: An overview of what keys are available to whom and when to support data recovery.

properly described in Figure 8 on page 17. Here we just abbreviate it as "Enc(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 is 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 that the server also puts 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 is directed to go through a procedure that is very similar to initial signup. The key difference is that she maintains the same name, email address and permissions instead of being treated as a new user by the system.

During this process she will generate a new personal keyset, (pk_C, sk_C), and she 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.[17]

What is not shown is that during this recovery signup Carol's client will generate a new Secret Key. Carol will pick a new Master Password which may be identical to her previous one. From her new Secret Key and her potentially new Master Password her client will generate a new MUK with which it will encrypt her new personal keyset.

[17]This would be a good time for Bob to confirm with Carol through some method other than email that it is indeed she who has reestablished her account.

Line 8   After Carol has created her new keyset and Bob confirms that he wishes to complete the recovery, the server will send Carol's new public key, pk_C, along with the copy of the vault key that has been encrypted with the Recovery Group' key, R_v. Recall that R_v was sent to the server when Carol first created the vault (5)

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 on to Carol, along with the encrypted data in the vault, d_{ve}.

There are several things to note about the process illustrated in Figure 17. Most importantly is that 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 also cancel the recovery if Carol successfully authenticated using her old credentials, thus somewhat reducing the opportunity for a malicious recovery without Carol noticing. None the less, 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.

How We Secure Data on Clients

Incomplete Content

We're sorry. This section of this document is not yet ready. Anything you see in this section is at most an outline of things to come.

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 that 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 applies to data when it is in transit.

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

2. TLS with best practices (Encryption, Data Integrity, Authenticity of server)

   Transport Layer Security (TLS), the successor of SSL, puts the "S" in "HTTPS". It encrypts data in transit and it authenticates the server so that the client knows to whom it is talking.

3. 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 is independent of TLS.

When discussing transport security it is useful to distinguish several different security notions; these are integrity, authenticity, and confidentiality.[18] "Confidentiality" means that the data remains secret; "authenticity" means that the parties in the data exchange are talking to whom they think they are; and data "integrity" means that the data transmitted cannot be tampered with. Tampering includes not only changing the contents of a particular message, but also preventing a message from

[18]When discussing information security the abbreviation "CIA" is often used to refer to Confidentiality, Integrity, and *Availability*. But when considering data transport security, authenticity plays a major role. In neither case should be abbreviation be confused with the well-known institution: the Culinary Institute of America.

getting to the recipient or injecting a message into the conversation that the authorized sender never sent.

Because parts of systems can fail, it is useful to design the overall system so that a failure in one part does not result in failure. This approach is often called **defense in depth**.

As summarized in Table 4, each encryption layer is independent of the others. If one fails, the others remain in place (though see Crypto over HTTPS 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. TLS, as it is typically used, authenticates the server but does not authenticate the client.

	SRP+GCM	TLS	At-rest encryption
Confidentiality	✓	✓	✓
Data integrity	✓	✓	✓
Server Authenticity	✓	✓	✗
Client Authenticity	✓	✗	✗

Table 4: *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.

Data at rest

Your 1Password data is always encrypted when it is stored anywhere[19], whether on your computer or on our servers, and it is encrypted with keys that are themselves encrypted with keys that are derived from your Master Password and Secret Key. Even if there were no other mechanisms to provide data confidentiality and integrity for the data that does reach the recipient, 1Password's at-rest encryption sufficiently provides both.

However, because it is designed for stored data, this layer of data encryption does not ensure that messages can't go missing or that older data is replayed. It also does not authenticate the communication channel.

[19]Decrypted Documents may be written to your own device's disk temporarily after you have opened them.

TLS

TLS is what 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.

Neither certificate pinning nor DNSSec have been implemented, but the certificate details for the *.1password.com certificate are listed in Figure 18.

DNS Names	`*.1password.com`, `DNS:1password.com`
Serial Number	`11:21:0d:96:4e:5e:39:39:12:40:fc:03:d3:22:40:8b:54:3f`
Key ID	`EF:F4:C6:1B:C7:AF:5B:3E:07:B3:22:AC:FE:D3:96:54:8B:9A:AF:EB`
Expires	Feb 18 20:09:50 2020 GMT
Issuer CN	AlphaSSL CA - SHA256 - G2
SHA1 Fingerprint	`53:76:A9:CB:26:56:21:D3:B0:78:B9:38:9A:18:4D:80:31:32:04:E0`

Figure 18: `*.1password.com` certificate details

The main 1Password website `https://1password.com` now uses an Extended Validation certificate for identity verification.

Our transport security

Our use of Secure Remote Password authentication between the client and the server provides mutual authentication. Both the server and the client will know that they are talking to whom they think they are.

This is in addition to the server authentication provided by TLS. Thus, if TLS fails in some instances to provide proper authentication, the Secure Remote Password still provides authentication.

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

Protecting Web Clients

See Crypto over HTTPS in Beware of the Leopard.

Server Infrastructure

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 which are 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 it is this relationship which determines membership within an account.

These relationships — users to accounts, accounts to vaults, vaults to items — do not 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 which 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 the `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.

Our Principle 3 states that 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 an `encrypted_by` field which references the `uuid` field in the `archived_vault_keys` table. The `enc_overview` and `enc_details` fields in the `vault_items` fields are encrypted with the key contained in the `enc_vault_key` field, which is itself encrypted.

- An `archived_vault_keys` entry is located using the `vault_id` field which equals the `vault_id` field in the `user_vaults` table for the desired user's `user_id`. The `enc_vault_key` field is encrypted with the user's public key and may only be decrypted with the user's private key.

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

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

With the hard work of the malicious database administrator, the user may have access to a `user_vaults` table entry which has the correct references, but since 1Password never has a copy of the unencrypted vault key, it is 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 would not have the private key needed to decrypt the encrypted vault key.

Our Principle 2 states that 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, our Principle 4 states that our design should be open for review. While we hope that our database administrators don't become malicious, we have provided all of the information needed to grant unauthorized access to encrypted items knowing that they will still be 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.

How is your data stored

1Password For Teams presently stores all database information using
an Amazon Web Services Aurora database instance. The Amazon Au-
rora service provides a MySQL compatible SQL relational database. Au-
rora 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 row and columns [20], with various indices de-
fined to improve performance.

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

The tables are listed as follows −

[20]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 does not possess.

- `accounts` - Table which contains registered teams, which originated
  from an initial signup request, approval, and registration. This ta-
  ble 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.

- `devices` - Table which contains a list of devices used by the user.
  The tables includes information for performing TOTP functions which
  are as-yet unimplemented. This table includes the cleartext last au-
  thentication IP address (`last_auth_ip`), client name (`client_name`),
  version (`client_version`), make (`name`), model (`model`), operat-
  ing system name (`os_name`) and version (`os_version`), and web
  client user agent string (`user_agent`).

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

- `signups` - Table which contains user requests to use the 1Password
  server. This table includes the cleartext team name (`name`) and email

address of the requester (`email`).

- `users` - Table which 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 all 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 which are subject to disclosure in the event of a data breach or required disclosure are:

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

- IP addresses used by devices

- TOTP 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 address and any custom avatar file names.

How are servers deployed and managed

Incomplete Content

We're sorry. This section of this document is not yet ready. Anything you see in this section is at most an outline of things to come.

Appendix A
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*."[1]

[1] Adams, *The hitchhiker's guide to the galaxy*.

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

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 is 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 Master Password and Secret Key. Likewise authentication in the web-client involves the same zero knowledge authentication scheme described in A modern approach to authentication.

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

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

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

3. The web client runs in a very hostile environment: the web browser. Some attacks on the browser (such as a malicious extension) may be able to capture user secrets. This is discussed further in Crypto in the browser.

4. Without the web-client users would only enter their Master Passwords into native clients and so would be less vulnerable to phishing attacks.

5. 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:
1. Use (code signed) native clients as much as possible.
2. Keep browser software up to date
3. Create a specific browser profile for using the web-client
4. Pay close attention to browser security warnings
5. Use on trusted network
6. Manually check certificates

Our mitigations include:
1. Use the most recent Transport Layer Security (TLS) version
2. Don't support weak cipher suites (so avoiding many downgrade attacks)
3. Use of safe JavaScript constructions.
4. Use HTTP Strict Transport Security (HSTS) (so avoiding HTTPS to HTTP downgrade attacks)
5. Pin Certificates (not yet implemented)

Pinning

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

Crypto in the browser

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

1. 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.

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

3. We have a strictly limited ability to use security features of the operating system when operating within the browser. See Locally exposed Secret Keys for how this limits the tools available for protecting Secret Keys when stored locally.

4. There is a paucity of efficient cryptographic functions available to run in JavaScript. As a consequence, the WebCrypto facilities available in the browsers that 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.

Recovery Group powers

From a cryptographic point of view, the members of a Recovery Group have access to all of the vault keys in that group. 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 the resources an attacker will put into acquiring it, members of the Recovery Group and their computers may be subject to targeted attacks.

> **!** Members of the Recovery Group must keep their systems secure and must be selected with care.

No public key verification

At present there is no practical method[2] for a user to verify that the public key they are 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 (MɪᴛM) 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.

This is discussed in greater detail in Verifying public keys.

Limited re-encryption secrecy

Revocation

Removing someone from a vault, group, or team is not 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 not 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 in that if they find a way to obtain the encrypted vault data. This is illustrated in Story 9. Note that this requires that the attacker both plan ahead and somehow acquire updated data.

[2] An impractical method for 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.

MɪᴛM: A Man in the Middle attack has Alice believing that she is encrypting data to Bob, while she is actually encrypting her data to a malicious actor who then re-encrypts the data to Bob. The typical defenses such an attack is for Alice and Bob to manually verify that they are 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..

Story 9: 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 he stores that copy separately from 1Password.

Wednesday Mr. Talk is discovered stealing Patty's toys and so is expelled from the vault (and from the team).

Thursday Patty updates the Squirrel Watchers vault with the new hidding 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 then is able to see the hiding place that Patty added on Thursday.

Mr. Talk needed to acquire a copy of the encrypted data that the server would no longer give him to launch the attack, and he needed to anticipate being fired.

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 is 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 will not be able to decrypt the data in the new vault no matter what encrypted data they gain access to.

Master Password changes don't change keysets

A change of Master Password or Secret Key does not create a new personal keyset; it only changes the Master Unlock Key (MUK) with which the personal keyset is encrypted. Thus an attacker who gains access to a victim's old personal keyset can decrypt it with an old Master Password and old Secret Key and use that to decrypt data that has been created by the victim after the change of the Master Password.

Your mitigations

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

Policy enforcement mechanisms not always clear to user

Readers of this document will understand from Access control enforcement that some controls (such as 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 that they do have use access to). The security properties of those differ enormously. In particular, it is 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 is a potential for them to get an incorrect impression of the security consequences of their decisions.

Malicious client

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

Vulnerability of server data

It should be assumed that governments, whether through law enforcement demands or other means, may gain access to all of the data that we have or that 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 Master Password and to keep your Secret Key secure.

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

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. Once malware running on a system has sufficient power, there is 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.

Locally exposed Secret Keys

Once 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 MUK it cannot be encrypted by the same MUK. Although lightly obfuscated, the Secret Key is stored on the local device unencrypted. Where possible, the Secret Key will be put into something provided by your system for storing authentication secrets. For 1Password for Mac and 1Password for iOS that will use the iOS and OS X keychains respectively. But when 1Password has been used from a web browser, the Secret Key is stored in the browser's local data store, a fairly exposed location.

Recall that the Secret Key is designed so that an attacker will not be in a position to launch an offline password guessing attack if she captures data from the server alone. It does succeed at that goal, but in the current version, our ability to protect the Secret Key on your computer is limited by the tools available to that particular client.

Revealing who is registered

If Oscar suspects that `alice@company.example` is a registered user in a particular Team or Family it is possible for him to submit requests to

our server which would allow him to confirm that 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 is only a mechanism that confirms whether a particular user is or isn't registered. Oscar must first make his guess and test that guess.

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

Use of email

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

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 that even if our authentication system is compromised it would never provide an attacker the means to learn anything about the secrets needed to decrypt someone's data.

We use Secure Remote Password (SRP) as our password-authenticated key exchange (PAKE) to achive the authentication goals set on in Figure 7 on page 15. With SRP, the client is able to compute from a password (and a few other things) a number that is imaginatively called x. This secret x is never transmitted.

Registration

The client computes x from the user's Master Password and Secret Key and from some non-secret information as described in With a strong KDF.

During first registration, the client will compute from x a verifier, v, which will be sent to the server during initial registration.

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 is sent only during initial registration.

The verifier v cannot be used directly to compute either x or the password that was used to generate x. However, it is 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.

Sign-in

The client will be able to compute *x* from the Master Password, Secret Key, and salt as described in the section on Key derivation.[1] 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

[1] The client may locally store *x* in a way that is encrypted with keys that depend on the Master Unlock Key (MUK) instead of recalculating it afresh each time.

1. Proves to the server that the client has the *x* associated with *v*.

2. Proves to the client that the server has the *v* associated with *x*,

3. Lets 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 that the client and the server present to each other is different each time, so one session cannot be replayed.

With a strong KDF

The standards documents describing 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 The math of SRP.

$$x \leftarrow \mathrm{SHA1}(s \| \mathrm{SHA1}(I \| " : " \| P))$$
$$v \leftarrow g^x \mod N$$

Figure B.1: Deriving *x* and *v* as given in RFC5054.

Although it is infeasible to compute *P* from *x* or to compute *x* from *v*, it is 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 does obtain *v* (which is stored on our server and is transmitted during initial registrations).

Infeasible: Effectively impossible. It is not *technically* impossible for a single monkey placed in front 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.

1. We use 2SKD with the completely random Secret Key as one of the secrets in deriving x. Password cracking is not a remotely feasible approach for an attacker without the Secret Key.
2. We use a slower key derivation function for deriving x than the one shown in Figure B.1, so that even if an attacker obtains both v and the user's Secret Key, each guess is computationally expensive.
3. We encourage the use of strong Master Passwords. Thus an attacker who has both v and the Secret Key will need to make a very large number of guesses.

For a discussion of (1) see Secret Key. The other 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,[2] the standard does not insist on it. Indeed, RFC 5054 refers to RFC 2954[3] §3.1 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 Key derivation.

The math of srp

Math background

The client will have its derived secret x, and the server will have its verifier, v. The mathematics that allows for the client and server to mutually authenticate and to 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 DLP is that if we pick N and g appropriately in equation B.2

$$v = g^x \pmod{N} \tag{B.2}$$

[2]Taylor et al., *Using the Secure Remote Password (SRP) Protocol for TLS Authentication*.

[3]T. Wu, *The SRP Authentication and Key Exchange System*.

it is easy (for a computer) to calculate *v* when given *x*, but it is infeasible to compute *x* when given *v*.

Calculating *x* from *v* (given *g* and *N*) is computing the **discrete logarithm** of *v*. To ensure that 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.[4] Given current and anticipated computing power, *N* should be at least 2048 bits.

[4]Kivinen and Kojo, *More Modular Exponential (MODP) Diffie-Hellman groups for Internet Key Exchange (IKE)*.

Diffie-Hellman key exchange

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

Assuming an appropriate *N* and *g*, Alice will not be able to determine Bob's secret exponent *b*, and Bob will not be able to determine Alice's secret exponent *a*. No one listening in − even with full knowledge of *g*, *N*, *A*, and *B* − will be able to determine *a* or *b*.[5]

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

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

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

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

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

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

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

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

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.

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 is only after they had 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 is her work in number theory that plays a role in cryptography today.

[5]There are numerous mathematical assumptions behind the claim that it is infeasible to determine *a* from *A*. Mathematicians are confident that some of the things involved are "hard" to compute but lack full mathematical proof. There are also some physical assumptions behind the security claims. We know that the relevant computations we would like to be hard are not hard using large quantum computers of a certain sort. We are assuming, with some justification, that constructing the appropriate sort of quantum computer is beyond anyone's reach for at least a decade. We do 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.

All of the secrets used and derived during DHE are ephemeral. They are created for the individual session alone. Alice will create a new *a* for each session; Bob will create a new *b* for each session; and 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.

Authenticated key exchange

DHE, as described in the previous section, allows Alice and Bob to agree on an encryption key for their communication. It does not, however, include a mechanism by which either Alice or Bob can prove to the other that 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 that she is the same "Alice" he has corresponded with previously, she needs to hold (or regenerate) a long term secret. At the same time we do not want to transmit any secrets during authentication.

SRP builds upon 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 is the only time a secret is transmitted.

As described in detail in Key derivation, *x* is derived from a Master 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 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.[6] The server will calculate a raw *S* as

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

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 that this is the only feasible way for the values to be the same is left for advanced texts.)
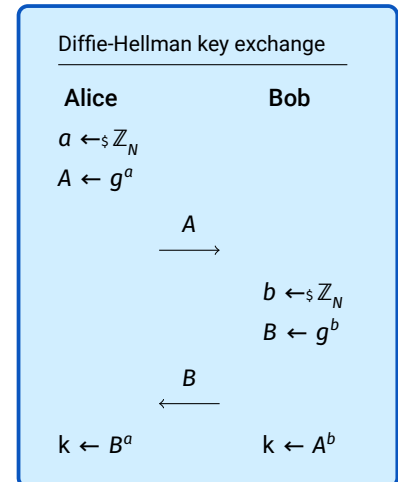

Figure B.3: In Diffie-Hellman key exchange, *N* and *g* are already known to all parties and all exponentiation is done mod *N*. The form given here is somewhat less general than it could be in order to avoid having to introduce more notation and abstractions.

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

Appendix C
Strengthening User Master Passwords

> **Incomplete Content**
>
> We're sorry. This section of this document is not yet ready. Anything you see in this section is at most an outline of things to come.

We have four mechanisms that make it more difficult for an attacker to acquire a Master Password.

1. Master Password cannot be guessed at in absence of the Secret Key.

2. Master Password is never transmitted

3. Encourage use of strong Master Password

4. Slow hashing of Master Password during key derivation.

Methods 1 and 2 are discussed in detail elsewhere, so this appendix will elaborate on 3 and 4.

Some words about strong Master Passwords

Master Passwords need to be both strong and usable. People will not do what is impossible of them, and any security design that expects the impossible of people is bound to fail.

Slow hashing

Appendix D

Securing Documents

> **Incomplete Content**
>
> We're sorry. This section of this document is not yet ready. Anything you see in this section is at most an outline of things to come.

- Documents have their own keys which are encrypted with the vault key.

- Decrypted documents may live in cache files in the local file system.

Appendix E
Verifying public keys

"Key verification is a weak point in public-key cryptography"[1]

At present there is no robust method for a user to verify that the public key they are encrypting data to belongs to their intended recipient.[2] 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 (MɪᴛM) 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 10 illustrates what might happen in the case of such an attack during vault sharing.

[1] Free Software Foundation, *The GNU Privacy Handbook*.

[2] The role of public key encryption in 1Password is described in How Vault Items Are Securely Shared and in Restoring a User's Access to a Vault.

Story 10: Mr. Talk is the cat in the middle

Molly (a dog) joins a team, and as she does, she generates public key pair. Let's say that the public key exponent 17 and public modulus 4171: $pk_M = (17; 4171)$. (Of course in the actual system that modulus would be hundreds of digits long.) Only Molly has access to the corresponding private part, $d_M = 2609$. When Patty (another dog) encrypts something using (17; 4171) only Molly, with her knowledge that d_M is 2609 can decrypt it.

But now suppose that Mr. Talk (the neighbor's cat) has taken over control of the 1Password server and database. Mr. Talk creates another public key, $pk_T = (17; 4183)$. Because Mr. Talk created that key, he knows that 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 pub-

lic key. But Mr. Talk, now in control of the server, doesn't send her Molly's real public key; instead he sends the fake public key that 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 that 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 that anything went wrong.

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

The use of plain RSA (and small numbers) in Story 10 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 out the math of the story recall that $d = e^{-1}$ mod $\phi(N)$; that for encrypting a message $c = e^m$ mod N; and that decrypting a ciphertext $m = c^d$ mod N. In our example $\phi(4157) = (43 - 1)(97 - 1) = 4032$, and $\phi(4183) = (47 - 1)(89 - 1) = 4048$.

For simplicity, story 10 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.

Types of defenses

The kind of problem we describe here is notoriously difficult to address, and it is fair to say that 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.

Trust hierarchy

The first defense requires everyone with a public key to prove that 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 that their key is theirs. That third party cannot be AgileBits or the 1Password server – the goal is to defend against a MɪᴛM 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 to publish it.

User-to-user verification

The second sort of approach is to enable users to verify keys themselves. They need to perform that verification over a communication channel that is 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 is 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[3] has confirmed that 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 is research that we are closely following.

[3]Dechand et al., "An Empirical Study of Textual Key-Fingerprint Representations".

The difficulty for users with verifying keys via fingerprints isn't just the technicalities of the fingerprint itself, but in understanding what they are 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."[4]

[4]Vaziripour, J. Wu, O'Neill, et al., "Action Needed! Helping Users Find and Complete the Authentication Ceremony in Signal".

Users may need to understand that

1. Fingerprints are not secret

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

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

4. That communication channel must be different than the communication system the user is trying to establish.

The developers of Signal, a well-respected secure messaging system, summarized some difficulties with fingerprints:[5]

[5]Marlinspike, *Safety number updates*.

- 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 are instructed to do so (in a laboratory setting) they often do not complete the process successfully.[6]

[6]Vaziripour, J. Wu, O'Neill, et al., "Is that you, Alice? A Usability Study of the Authentication Ceremony of Secure Messaging Applications"; Vaziripour, J. Wu, O'Neill, et al., "Action Needed! Helping Users Find and Complete the Authentication Ceremony in Signal".

The problem remains

We are aware of the threats posed by MɪᴛM, and users should be aware of those, too. We will continue to look for solutions, but we are 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.

Glossary

Account Key

Previous name for the Secret Key. *see* Secret Key

Advanced Encryption Standard (AES)

Probably the best studied and most widely used symmetric block cipher. 17, 45

authentication

Authentication is 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. 47, 63

authenticity

Data authenticity involves knowing who created or sent a message. The typical mechanisms used for this with respect to data are often the same as those used to protect data integrity; however, some authentication process may be necessary prior to sending the data. 45

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. 27

Chosen Ciphertext Attack (CCA)

A class of attacks in which the attacker modifies encrypted traffic in specific ways and may learn plaintext by observing how the decryption fails. 17

confidentiality

Data confidentiality involves keeping the data secret. Typically this is achieved by encrypting the data. 45

Cryptographically Secure Pseudo-Random Number Generator (CSPRNG)

A random number generator whose output is indistinguishable from truly random. Do not worry about the "pseudo" in the name. A CSPRNG is fully appropriate for generating cryptographic keys. 23

Diffie-Hellman key exchange (DHE)

Diffie-Hellman key exchange is an application of the discrete logarithm problem (DLP) to provide a way of for parties to decide upon a secret key without revealing any secrets during the communication. It is named after Witfield Diffie and Martin Hellman who published it in 1976. 61, 63

discrete logarithm problem (DLP)

If $y \equiv g^x \mod p$ (for a carefully chosen p and some other conditions) it is possible to perform exponentiation to compute y from the other variables, but it is thought to be infeasible to compute x from y. Computing x from y (and the other parameters) is reversing the exponentiation and so is taking a logarithm. 61

Ellipic Curve Cryptography (ECC)

A public key encryption system that is able to work with much smaller keys than are used for other public key systems. 19

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.[7]. 24, 68

[7]National Institute of Standards and Technology, *FIPS PUB 186-4: Digital Signature Standard (DSS)*.

Emergency Kit

A piece of paper that contains your Secret Key, Master Password, and details about your account. Your Emergency Kit should be stored in a secure place and used if you forget your Master Password or lose your Secret Key. 11, 12

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. 13, 15, 16, 52, 59

Galois Counter Mode (GCM)

An authenticated encryption mode for use with block ciphers. 17, 45

Hash-based Key Derivation Function (HKDF)

A key derivation function that uses HMAC for key extraction and expansion.[8] Unlike PBKDF2, it is not designed for password strengthening. 18, 26

[8]Krawczyk, *Cryptographic Extraction and Key Derivation: The HKDF Scheme*.

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. 46, 53

infeasible

Effectively impossible. It is not *technically* impossible for a single monkey placed in front 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. 60, 62

integrity

Data integrity involves preventing or detecting tampering with the data. Typically this is done through authenticated encryption or message authentication. 45

JSON Web Key (JWK)

A format for describing and storing cryptographic keys defined in RFC 7517[9]. 18, 26, 30

[9]Jones, *JSON Web Key (JWK)*.

key encryption key (KEK)

An encryption key that is used for the sole purpose of encrypting another cryptographic key. 17

Key Set

How collections of keys and their metadata are organized within 1Password for Teams. 18–20

Man in the Middle (MitM)

A Man in the Middle attack has Alice believing that she is encrypting data to Bob, while she is actually encrypting her data to a malicious actor who then re-encrypts the data to Bob. The typical defenses such an attack is for Alice and Bob to manually verify that they are 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.. 55, 66, 67, 69

Master Password

Your Master Password is something that you must remember and type when unlocking 1Password. It is never transmitted from your devices. 6, 9–14, 16–18, 24–26, 28, 29, 31, 33, 37–39, 42, 46, 52, 53, 56, 57, 59–61, 63, 64, 74

Master Unlock Key (MUK)

Key used to encrypt a user's private key. It is derived from the user's Master Password and Secret Key. 18, 24–28, 30, 31, 39, 42, 56, 57, 60

multi-factor authentication (MFA)

Requiring a combination of secrets (broadly speaking) such as a password or a cryptographic key held on a device to grant access to a resource. 16, 79

mutual authentication

Mutual authentication is a process in which all parties prove their identity to each other. 47, 73

P-256

A popular standard[10] elliptic curve and public parameters for elliptic curve public key cryptography, such as with Elliptic Curve Digital Signature Algorithm (ECDSA). When used appropriately, it provides 128-bit security.. 24

password-authenticated key exchange (PAKE)

Password-based key exchange protocol allows for a client and a 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. 15, 16, 59

Recovery Group

The 1Password for Teams Group that has a copy of the vault keys for vaults which may need to be recovered if Master Passwords or Secret Keys of vault members are lost. 19, 38–40, 42, 43, 54, 58

Recovery Group Member

A member of a Recovery Group. *see* Recovery Group

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 encryp-

[10]Turner et al., *Elliptic Curve Cryptography Subject Public Key Information*.

tion is unique. Salts are typically random and unique. 18, 24, 26, 27, 59, 60

Secret Key

A randomly generated user secret key that is created upon first signup. It is created and stored locally. Along with the user's Master Password it is 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 Master Password.

Previously known as the "Account Key". 9–14, 16–19, 23–29, 31, 33, 37–39, 42, 46, 52, 54, 56, 57, 59–61, 63, 64

Secure Remote Password (SRP)

The Secure Remote Password protocol is a method for both a client and a 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 are using Version 6[11] with a modified key derivation function. 6, 13, 15, 27, 28, 59, 60, 63

[11] Taylor et al., *Using the Secure Remote Password (SRP) Protocol for TLS Authentication*.

SRP-*x*

The client secret, *x*, used by the Secure Remote Password (SRP) protocol. It is derived from the user's Master Password and Account Key. 24, 27–30, 39

Transport Layer Security (TLS)

TLS is the successor to SSL. It's what puts the "S" in "HTTPS". 2, 14, 15, 29, 45–47, 53

two-secret key derivation (2SKD)

Two different secrets, each with their own security properties, are used in deriving encryption and authentication keys. In 1Password for Teams these are your Master Password (something you know) and your Secret Key (some high-entropy secret you have on your device). 6, 9–11, 13, 16, 24, 38, 60, 61, 79, 80

Unicode Normalization Form Compatibility Decomposition (NFKD)

A consistent normal form for Unicode characters that could otherwise be different sequences of bytes. 25

Universally Unique Identifier (UUID)

A large arbitrary identifier for an entity. No two entities in the universe should have the same UUID. 27–29

zero knowledge protocol

A zero knowledge protocol is a way for parties to make use of secrets
without revealing those secrets to each other. *see also* srp

Bibliography

Adams, Douglas. *The hitchhiker's guide to the galaxy*. 1st American ed. New York: Harmony Books, 1979.

AgileBits. *OPVault Design*. Aug. 28, 2015. URL: `https://support.1password.com/opvault-design/` (visited on 11/16/2015).

Dechand, Sergej et al. "An Empirical Study of Textual Key-Fingerprint Representations". In: *25th USENIX Security Symposium (USENIX Security 16)*. Austin, TX: USENIX Association, 2016, pp. 193–208. ISBN: 978-1-931971-32-4. URL: `https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/dechand`.

Free Software Foundation. *The GNU Privacy Handbook*. The Free Software Foundation. 1999. URL: `https://www.gnupg.org/gph/en/manual.html` (visited on 12/05/2017).

Goldberg, Jeffrey. *You have secrets; we don't: Why our data format is public*. English. AgileBits. May 6, 2013. URL: `https://blog.1password.com/you-have-secrets-we-dont-why-our-data-format-is-public/`.

Jones, M. *JSON Web Key (JWK)*. RFC 7517 (Proposed Standard). Internet Engineering Task Force, May 2015. URL: `http://www.ietf.org/rfc/rfc7517.txt`.

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

Knuth, Donald Ervin. *The TEXbook*. Reading, Mass.: Addison-Wesley, 1984. ISBN: 0201134489 (soft).

Krawczyk, Hugo. *Cryptographic Extraction and Key Derivation: The HKDF Scheme*. Cryptology ePrint Archive, Report 2010/264. 2010.

Marlinspike, Moxie. *Safety number updates*. Open Whisper Systems. Nov. 17, 2016. URL: `https://signal.org/blog/safety-number-updates/` (visited on 10/05/2017).

National Institute of Standards and Technology. *FIPS PUB 186-4: Digital Signature Standard (DSS)*. Gaithersburg, MD, USA: National Institute

for Standards and Technology, July 2013. URL: https://csrc.
nist.gov/publications/detail/fips/186/4/final.

Pornin, Thomas. *The MAKWA Password Hashing Function*. 2015. URL:
http://www.bolet.org/makwa/makwa-spec-20150422.
pdf.

Taylor, D. et al. *Using the Secure Remote Password (SRP) Protocol for
TLS Authentication*. RFC 5054 (Informational). Internet Engineering
Task Force, Nov. 2007. URL: http://www.ietf.org/rfc/rfc5054.
txt.

Turner, S. et al. *Elliptic Curve Cryptography Subject Public Key Informa-
tion*. RFC 5480 (Proposed Standard). Internet Engineering Task Force,
Mar. 2009. URL: http://www.ietf.org/rfc/rfc5480.txt.

Vaziripour, Elham, Justin Wu, Mark O'Neill, et al. "Is that you, Alice? A
Usability Study of the Authentication Ceremony of Secure Messaging
Applications". In: *Symposium on Usable Privacy and Security (SOUPS)*.
2017.

Vaziripour, Elham, Justin Wu, Mark O'Neill, et al. "Action Needed! Helping
Users Find and Complete the Authentication Ceremony in Signal". In:
Fourteenth Symposium on Usable Privacy and Security (SOUPS 2018).
Baltimore, MD: USENIX Association, 2018, pp. 47–62. ISBN: 978-1-
931971-45-4. URL: https://www.usenix.org/conference/
soups2018/presentation/vaziripour.

Wu, T. *The SRP Authentication and Key Exchange System*. RFC 2945
(Proposed Standard). Internet Engineering Task Force, Sept. 2000.
URL: http://www.ietf.org/rfc/rfc2945.txt.

Changelog

This describes the changes made to the document "1Password Security Design: Teams, Families and Individual Accounts"

[0.2.10] – 2019-01-12

Improved

- No proprietary fonts (in preparation for opening source)

[0.2.9] – 2018-12-30

Improved

- Fixed text citation in MitM appendix
- Less verbose bibliographic data in margins

[0.2.8] – 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

[0.2.7] – 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

[0.2.6] – 2017-04-11

The Account Key has been renamed "Secret Key"
    Substantially restructured presentation of authentication, including a new section that 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.

New

- New section on A modern approach to authentication

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.
- T$_E$Xnical changes to improve line breaking

[0.2.5] - 2017-02-20

Mostly typos, minor fixes. These are not the changes you are looking for.

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"

[0.2.4] - 2016-09-28

Details of account recovery have been added. Continue to fill in blanks, but not as many as hoped for over such a long period of time. New presentation of Two-Secret Key Derivation.

New

- Detailed section on Restoring a User's Access to a Vault
- AK and MP rewritten in terms of 2SKD

Improved

- Typography
- More detail in "Beware of Leopard"
- More math in SRP
- Many small improvements and expansions throughout the text

Fixed

- Many many typos and phrasing awkwardnesses.
- Colors are no longer emetic

[0.2.3] - 2015-11-27

Continue to expand the paper, completing sections which were not complete earlier.

New

- Documented how server-side data is stored and which keys are exposed.
- Expanded explanation of SRP authentication

Fixed

Numerous typos.

[0.2.2] - 2015-11-03

First public 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.

New

First public release.