End-to-End Encryption for Day One Sync

From the time that we started designing end-to-end encryption for Day One Sync, we've planned to publish the technical details of our implementation. Recently, we were inspired by a similar disclosure by OmniFocus (see it here). Now, with our design reaching stability and implementation well underway, we're ready to do the same.

We'll start by explaining what we're trying to accomplish with end-to-end encryption. Then we'll briefly review the current state of Day One Sync security so you can see what protections are already in place. The remainder of the paper will provide the technical details of how we'll close the gaps.

Our implementation will receive a professional security audit, but we welcome public feedback too. You can comment by emailing security@dayoneapp.com.

Your personal journaling data

Personal journaling data consists of your entry content (text and images) and the various bits of life-record data that can be attached to a Day One entry.[1]

Your personal journaling data belongs to you and is yours to control. That is our guiding principle.

Some other kinds of synced data do not qualify as personal journaling data and are not encrypted end-to-end: the date and time of an entry and when it was edited; image type and dimensions; technical information about the devices and platforms you use with Day One; and statistics such as the number of journals, entries, and images. We use this data only for internal purposes (customer support, sync functionality, business metrics, etc.) and treat it as confidential.

Our goals

Privacy: Your personal journaling data can only be read by you. You explicitly authorize the devices that can decrypt it. Even someone with full access to the sync infrastructure (servers, network, database, data storage, etc.) can't read it or secretly tamper with it.

Security: We use standard encryption technologies that are considered very strong. We can evolve and strengthen our encryption measures over time as expert recommendations change, while preserving compatibility. We use cryptographic keys instead of passwords to secure your data because passwords are notoriously susceptible to attack. Even in the unlikely event that an

[1] Examples include image metadata, location, weather, tags, step counts, and so forth.

encryption key is compromised, it's easy to replace it with a new key, and the amount of data potentially exposed is minimized.

Functionality: Services such as our IFTTT channel can create entries in your journal without being able to read them (or anything else in your journal) thereafter. The other goals of Day One Sync unrelated to security (efficiency, ease of use, no data loss, etc.) continue to be met.

The goals do NOT include:

- Protecting the journal data at rest on your device. The built-in sandbox model, disk encryption, and your device's passcode/TouchID/password features provide the access control needed to keep it safe. We specifically recommend against “jailbreaking” your iOS devices, as this weakens many of the data security measures Apple provides.
- We can't absolutely prevent data loss. If you lose the key to your encrypted data, we can't decrypt it for you. (But you probably have a backup of your data on your device.)

Note: Unrelated to encryption and sync, but worth mentioning here, is that we don't want to hold your data hostage or lock you into our services exclusively. Day One provides several export formats including plain text, PDF, and JSON, to allow you to store or process your journal data as you see fit.

Sync without end-to-end encryption

Even in versions of Day One without end-to-end encryption, we have measures in place to protect your personal journaling data. Synced journal data is encrypted during transfer between the device and the Day One servers[2], and also encrypted when written to disk storage.[3] It is unencrypted when being processed on the sync server and handled by the database.

Only a small number of Day One engineers have access to the servers and database. Several security measures protect this access.[4] We have also taken measures to prevent those with access from inadvertently viewing actual journal content.[5] But this is still less than the level of privacy and security we want your synced data to have: we want to make it impossible for us or anyone else to have unauthorized access to your journal, if you choose.

---

[2] This encryption is achieved by requiring Transport Layer Security (TLS) for all Sync traffic.
[3] This encryption is provided transparently by the Amazon Web Services (AWS) storage infrastructure, using S3 and EBS encryption.
[4] These include: SSH authentication restricted to public key only; IP whitelist firewall rules; two-factor authentication; strong password requirements; and separate dev/test and production AWS accounts.
[5] Day One servers Base64-encode the text of each entry before it is written to the database, so that an engineer can view an entry record without seeing what the user wrote in it. This weak form of privacy protection is intended only as a safeguard against inadvertent viewing, not intentional access which can only be prevented by end-to-end encryption.

End-to-end encryption

Our goal for *end-to-end encryption* is that (a) your personal journaling data is encrypted on your device before it is synced to the Day One servers, (b) it can only be decrypted by another synced device that has your key, and (c) you never have to share your private key with Day One or anyone else in order to use Day One Sync.

Important terminology:

- A *symmetric key* can be used both to encrypt and decrypt data. Encryption algorithms based on symmetric keys tend to be fast and capable of processing large amounts of data.
- An *asymmetric key pair* consists of a *public key* that is used to encrypt data but cannot decrypt it, and a *private key* that decrypts what the public key encrypted. Encryption algorithms based on asymmetric keys are best suited for processing data of limited size[6].
- A *signature* is used to verify the integrity of the data being synced, in order to prevent tampering. A private key is used to sign the data, and the public key is used to verify the signature.

General design

We rely on a hybrid encryption approach, where symmetric and asymmetric encryption work together. This happens at several levels of your data: the entry, the journal, and your user account.

Entry text and life-record data for a single entry are encrypted with a randomly generated symmetric key called the **entry key**. The same key is used for decryption. Each entry has its own independent entry key. Likewise, images are encrypted and decrypted with their own randomly generated, independent, symmetric **image key**.

Each entry or image key ("content key") is secured by the asymmetric **journal key pair**. The public journal key is used to encrypt the content key. Both the server and device can use the public journal key to create new content and encrypt its key. However, only the device knows the private journal key, so only it can decrypt a content key. Consequently, the server can add new entries and images to the journal, but cannot read or update them.

Each journal has its own list of journal key pairs called the **journal vault**. The newest key in the vault is designated as the active key pair, which is used to encrypt and sign new entries. The rest are "retired" and used only for decrypting and verifying entries that were previously encrypted with that key pair.

---

[6] The 2048-bit asymmetric keys we are using can encrypt only 256 bytes at a time.

The vault for a journal is secured with a randomly generated symmetric **vault key**. This key encrypts and decrypts the private keys in the vault. (The public keys remain unencrypted for server-side use.) When a new journal key is added, the vault key must also be changed. Each journal has its own independent vault and associated vault key.

Each journal's vault key is secured using an asymmetric **user key pair**. Each user account has its own independent user key, which is always generated on your device. The public key is transmitted and stored on the sync server with your user account, but the private key is only known to the device. The user public key is used by the server to verify signatures used in updating other keys.

The user private key is protected by the **user master key**. This alphanumeric key is the only key ever presented to the user. Like all the other keys, it is generated on the device, but the user master key is never sent to the server—instead, the user is responsible for entering this key on each device that will have access to their account. The user master key is used to generate a symmetric key, which is used to encrypt the userprivate key.

Trust and verification

Entries can be created either on your device or via server-side processes such as IFTTT. But only your device can read and update an entry after its creation.

If the entry key was generated by the server, it is not trusted for use in future updates made on your device. Otherwise, an attacker with server access could use that key to read the updated content. We use cryptographic signatures to accomplish this.

- **When you create an entry on your device**, it generates a signature of the encrypted entry key using the private journal key. This signature is stored alongside the encrypted entry key. When verified, it proves that the entry key was generated by a device with access to the private journal key, and therefore can be trusted.
- **When a server-side process such as IFTTT creates an entry**, it encrypts the entry key with the public journal key. But because it doesn't have access to the journal private key, it cannot sign the encrypted entry key. The absence of a signature is a signal to your device that the entry key needs to be replaced with a new one, so that future updates to that entry can't be read by the server.[7]

The encrypted journal vault key is signed with the uesr master private key, to allow verification that the key and the vault it encrypts are trusted. This prevents an attacker from secretly replacing a journal vault and key with their own.

[7] In practice the server will discard the plaintext content key after it's been used to encrypt the new content. But using signatures means you don't have to rely on the server to do the right thing.

Implications of this design

This chain of encryption starts at the entry and image data level, and access is controlled all the way to the user master key. Let's see how this works:

1. To decrypt the data in an entry, you must have the decrypted **entry key**. Likewise, to decrypt an image, you must have the decrypted **image key**.
2. To decrypt the entry or image key, you must have the **journal private key** which is encrypted in the journal vault.
3. To decrypt the journal private key from the vault you must have the decrypted **vault key** for that journal.
4. To decrypt the journal vault key you must have the decrypted **user private key**.
5. To decrypt the user private key, you must have the **user master key**, which is controlled by you.

In the unlikely event that a key is somehow compromised, here is what data might be exposed and how to recover from such a situation:

- If an entry key is compromised, only the data associated with that entry can be decrypted; other entries are still safe. To recover, the key can be replaced with a new one, requiring only the data for that entry to be re-encrypted. The same is true of images and image keys.
- If a journal private key is compromised, only the entries and images in that journal that were encrypted using that key pair can be decrypted, while other entries and images in that journal and all other journals are still safe. (An attacker would still need access to the entry records and images themselves, in addition to the journal private key.) To recover, a new journal key can be generated and added to the journal vault so that future updates are made using the new, non-compromised key.
- If a journal vault key is compromised, all of the content in that journal can be decrypted (if the attacker has access to the records), but all other journals are still safe. To recover, a new journal key can be added to the vault and encrypted with a new journal vault key, so that future updates to the journal can't be read by the attacker.
- If your user private key or your user master key is compromised, all of the content in all of your journals can be decrypted (if the attacker has access to all of the journal and entry records and images), but all other users' data is still safe. To recover, a new user master key pair can be generated and used to re-secure the journal vault as above. The new private key would also need to be distributed to your other devices.

Additional details

- We use the AES256-GCM cipher for symmetric encryption of journal vaults and personal journal data because it can encrypt potentially large amounts of data, allows

authentication of data, and is regarded as efficient and strong. Initialization vector (IV) size is 12 bytes and authentication tag size is 16 bytes (128 bits).

- All symmetric keys and IVs in our design are randomly generated using the most secure (high-entropy) randomness sources available. GCM cipher security requires that the same IV is never used twice with the same key. Generating a random 12-byte value for each encryption operation has a near-zero probability of repeating a prior IV, and since each entry uses a separate encryption key, the chance of reusing the same key and IV together is negligible.
- We use RSA for asymmetric encryption of symmetric keys and for signing and verifying data. We use a 2048-bit modulus size, with OAEP padding using SHA-1 for MGF1.
- The user master key is composed of 31 randomly selected alphanumeric characters, drawn from a pool of 30 characters, resulting in a bit over 2^{152} possible master keys. For encryption purposes, the user master key is converted into a 256-bit symmetric key using the PBKDF2 algorithm, using 100000 rounds of SHA256, and with the user account ID as a salt.
- When an entry is edited in the Day One app, we want to ensure that only devices with access to the current journal private key can access the new revision of the entry. To that end, if the entry's symmetric key was secured using anything other than the current journal key, or if the associated signature is missing, then a new symmetric key is generated for that entry.
- If your device ever receives an update from the server containing an invalid signature, it will ignore that update.

Future work

Here are some things we're considering for the future, possibly as premium features:

- Shared encrypted journals: you could grant and revoke access to an encrypted journal to another Day One user, who would have full read/write functionality for entries in that journal during that time but no access to updates that happen after the access is revoked.

Comments and suggestions are welcome. Email security@dayoneapp.com.