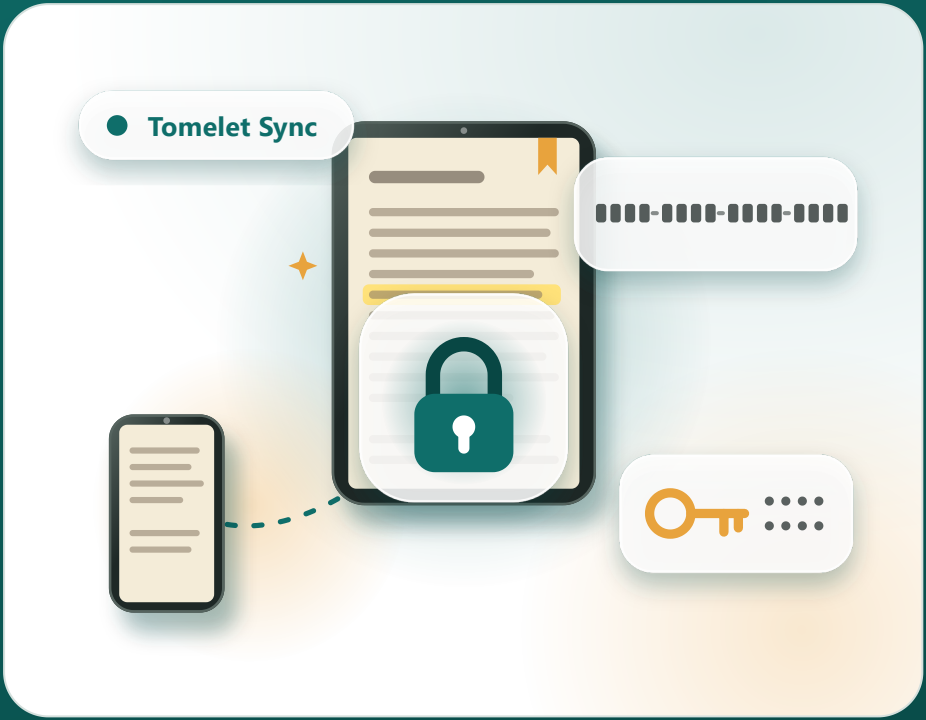
Tomelet

WHITE PAPER



Tomelet Sync

HOW TOMELET KEEPS YOUR LIBRARY PRIVATE ON EVERY DEVICE

Overview

Reading now moves between devices. A book started on a phone continues on a tablet in the evening and on a laptop at work, and the highlights, notes and handwriting gathered along the way become the most personal part of a library.

Most reading services store that library in a form the provider can read, which leaves a reader's thinking exposed to anyone who runs, breaches or compels the service. Tomelet Sync keeps a library in step across every device while keeping its contents unreadable to Tomelet. Its design rests on five ideas:

End-to-end encryption

Books, their details, highlights, notes, pen ink and reading positions are encrypted on the reader's device with AES-256-GCM before they are uploaded. Tomelet stores ciphertext it cannot open.

Device-held keys

The key to a library exists only on the reader's approved devices, and in a sealed form that only those devices and the reader's own Recovery Key can open.

Verified device approval

A new device joins a library only when an existing device approves it, after the reader compares a verification code shown on both screens.

A Recovery Key the reader holds

A 128-bit key, generated on the device and shown once, restores a library on a new device without any involvement from Tomelet.

Tamper-evident sync

Every change is authenticated and every sync request is signed by the device that sent it, so the server can store and order a library without being able to alter it.

Together, these give readers the convenience of cloud sync with the privacy of a library that stays under their control.

Contents

Overview 2

What is Tomelet Sync? 4

Benefits of Tomelet Sync 5

Challenges of private sync 6

- Sync done right 6

What does Tomelet Sync achieve? 7

- Trusted and untrusted components 8

How we designed Tomelet Sync 9

- The vault key and its seals 10
- Creating your vault 11
- Why a Recovery Key instead of a password? 11

Adding a device 12

- Adding a device with a verification code 13
- Why compare the code? 14
- Removing a device 14

Encrypting and protecting your library 15

What our servers see 16

Sign-in and sessions 17

Keys on your device 18

Conclusion 19

What is Tomelet Sync?

Tomelet is an ebook reader for Android, iPhone and iPad, Mac, Windows, Linux and the web. The reader itself is free, works offline and needs no account; everything it does happens on the device.

Tomelet Sync is the subscription that connects those devices. It carries a reader's whole library between them: the book files and covers, titles and authors, reading positions and history, highlights, notes, bookmarks, pen ink, shelves, tags and collections. Every one of them is encrypted on the device before it leaves.



Figure 1. *Tomelet Sync keeps one encrypted library in step across every platform Tomelet runs on.*

Benefits of Tomelet Sync

A reader who subscribes to Tomelet Sync keeps one private library across all of their devices. Here are a few of the moments where that matters most.

PICKING UP WHERE YOU LEFT OFF

A book opens on the next device at the page where the reader stopped, with every mark in place, whichever platform they reach for. The morning's chapter on a phone becomes the evening's chapter on a tablet without a second thought.

KEEPING YOUR THINKING WITH YOU

Highlights, written notes and handwritten ink are often worth more to a reader than the book itself. Tomelet Sync keeps them together with the book on every device, anchored to the words they belong to, so a note made on one screen reads correctly on another.

ONE LIBRARY ACROSS PLATFORMS

A reader who moves between an Android phone, an iPad and a Windows laptop keeps one library, instead of a separate collection on each. Books imported on one device are available on all of them.

RECOVERING FROM A LOST DEVICE

When a phone is lost or replaced, the library returns on the new device after an approval from another of the reader's devices, or with the reader's Recovery Key when no other device is at hand.

Challenges of private sync

Keeping a library in step across devices is a solved problem. Doing it without the provider being able to read the library raises four challenges.

1. KEEPING THE SERVER OUT OF THE LIBRARY

A sync service has to store, order and deliver a reader's data. If it can also read that data, the privacy of every library depends on the security of the service, the people who run it and the legal environment it operates in. A private design has to let the server do its work without ever giving it the means to read.

2. ADDING DEVICES SAFELY

When keys live only on a reader's devices, a new device has to receive them from somewhere. A design that simply trusts the server to introduce new devices would let a compromised server introduce itself, so the introduction has to be verified by the reader.

3. RECOVERY WITHOUT A BACK DOOR

Familiar "forgot password" flows work because the provider can reset access, which means the provider holds a key. Private sync needs a recovery path that belongs entirely to the reader.

4. EVERYDAY USABILITY

Strong encryption only protects readers who can use it. Setting up sync, adding a device and recovering a library have to feel as simple as signing in, with no technical knowledge required.

> **SYNC DONE RIGHT**
>
> These challenges shaped every decision in Tomelet Sync. Our goal was a service that readers can use without thinking about cryptography, built so that their library remains private even if our own servers are breached. This white paper sets out the engineering design and its implementation, so that readers, researchers and partners can see exactly how Tomelet Sync protects a library.

What does Tomelet Sync achieve?

Tomelet Sync answers those challenges with four design tenets. Each one holds for every library, on every platform, at every moment.

ZERO-KNOWLEDGE STORAGE

Tomelet's servers store, order and deliver a library as ciphertext. They never receive a key that can open it, so the contents of a library are protected from Tomelet, from anyone who breaches Tomelet and from anyone who compels Tomelet to hand data over.

DEVICE-HELD KEYS

Each library has its own vault key, made on the reader's first device. It travels to other devices only sealed to their own keys, after the reader approves them.

TAMPER EVIDENCE

Every change carries authentication that binds it to its place in the library, and every sync request carries the signature of the device that made it. A server that altered, replayed or reordered data would be detected.

LOCAL FIRST

A reader's library lives on their devices and keeps working without a connection. Sync is a private layer on top, so a library remains available on each device even when the service is not.

Trusted and untrusted components

A fundamental design principle of Tomelet Sync is that only components the reader controls ever hold the keys to their library. We treat the reader's approved devices as trusted: they run the Tomelet app the reader installed and keep their keys in the platform's secure storage. We treat everything else as untrusted, including our own servers, so that the design protects a library even if one of those components is breached, misconfigured or compelled.

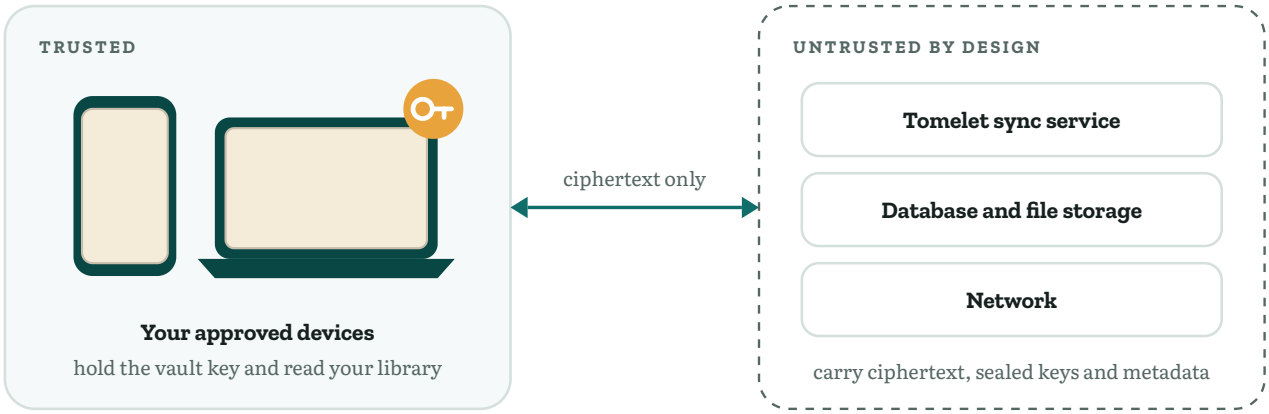


Figure 2. *Only the reader's approved devices can use the keys. Everything between them carries ciphertext.*

COMPONENT	HOW WE TREAT IT	WHAT IT HOLDS
Your approved devices	Trusted	The vault key, the device's own keys and your readable library.
Tomelet's sync service and database	Untrusted	Ciphertext, sealed keys, public keys and the account metadata described in this paper.
Cloud storage and network	Untrusted	Ciphertext, carried over TLS.
Sign-in providers	Trusted to confirm identity only	Your sign-in, never a key to your library.

How we designed Tomelet Sync

This section examines the architecture in depth. It begins with the cryptographic building blocks, then follows a library from the moment its vault is created to the moment a new device joins.

CRYPTOGRAPHIC BUILDING BLOCKS

Tomelet Sync is built entirely from standard, widely studied primitives, implemented in the apps with the open-source *cryptography* library for Dart and on the server with Node.js's built-in cryptography. Random values come from the operating system's secure random number generator.

PRIMITIVE	USED FOR
AES-256-GCM	Encrypting every change, every file and every sealed key, with a 96-bit nonce and a 128-bit authentication tag.
X25519	Sealing the vault key to one device, with a fresh one-time key for every seal.
HKDF-SHA256	Deriving each purpose's key from its parent, with a distinct label for every purpose.
Ed25519	Device signatures on sync requests, and the vault's signatures on device approvals.
SHA-256	Naming encrypted files by their content, and deriving the verification code.
PBKDF2-HMAC-SHA256	The optional complete-archive export, protected by a passphrase the reader chooses, at 600,000 iterations.

The vault key and its seals

Every library is protected by one **vault key**: 32 random bytes made on the reader's first device. The vault key never leaves a device in readable form. Around it sit the keys that let it travel safely:

- **Device keys:** Each installation of Tomelet has two key pairs of its own, an X25519 pair that the vault key is sealed to and an Ed25519 pair it signs requests with. The private halves stay in the device's secure storage.
- **Device seal:** To give a device the vault key, an approving device combines a fresh one-time X25519 key with the new device's public key, derives a sealing key with HKDF over the account, the key's identity, the device and both public keys, and encrypts the vault key with AES-256-GCM, binding the same details as authenticated data. Malformed or weak public keys are rejected.
- **Recovery seal:** The vault key is also sealed under a key derived from the reader's Recovery Key, so the Recovery Key alone can open it.
- **Vault authorization key:** An Ed25519 key derived from the vault key. The server keeps only its public half, which lets it confirm that a device approval came from someone holding the vault key without ever holding that key.
- **Change keys:** Each change a reader makes is encrypted under its own key, derived from the vault key and the change's identity.
- **File keys:** Each book file, cover and drawing of pen ink is encrypted under its own random 256-bit key, in 1 MiB chunks. That key travels only inside an encrypted change.

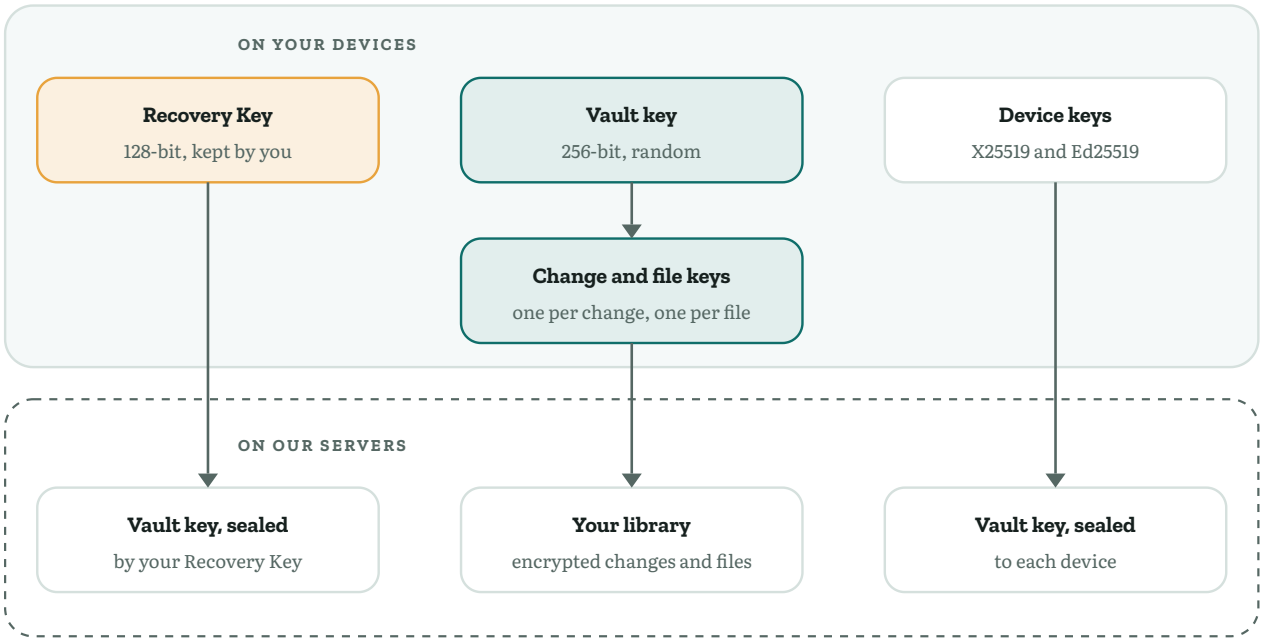


Figure 3. *The vault key is stored on our servers only in sealed form: once under the reader's Recovery Key and once for each approved device. The library is stored under change and file keys that only the vault key leads to.*

Creating your vault

A reader's vault is created the first time they turn on Tomelet Sync:

1. The device generates the vault key and a Recovery Key of 128 random bits.
2. It seals the vault key to its own device key and under the Recovery Key, and derives the vault authorization key.
3. It shows the Recovery Key once. The Continue button waits ten seconds, and the reader confirms they have kept the key by typing part of it back. If they copy it, the clipboard is cleared after 90 seconds.
4. It uploads only the sealed copies and the public half of the vault authorization key.

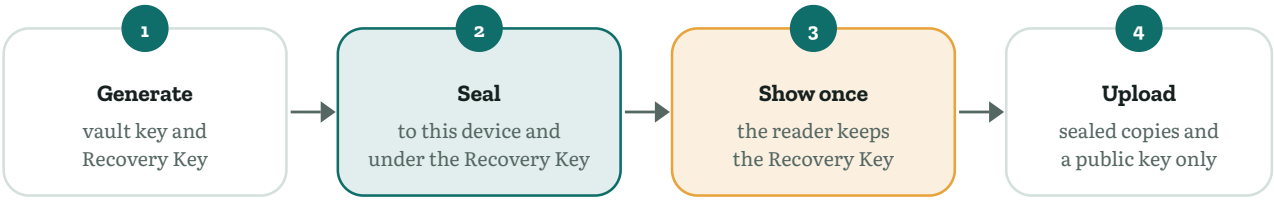


Figure 4. *A vault is created entirely on the reader's device. Only sealed copies of its key ever leave.*

The Recovery Key is written as **TR-** followed by 26 letters and digits and a two-character checksum, in groups of five, so a typing mistake is caught at once. Tomelet never receives the Recovery Key itself, which is why the reader is the only person who can use it.

> **WHY A RECOVERY KEY INSTEAD OF A PASSWORD?**
>
> A password is chosen by a person, and people choose words they can remember. Anyone who obtained a key sealed under a password could try billions of guesses offline until one fitted. The Recovery Key is generated by the device from 128 random bits, so there is nothing to guess: a search through every possible key is far beyond any computer. It also means a reader never has to invent or remember a vault password.
>
> To recover on a new device with no other device available, the reader signs in and enters the Recovery Key; the device opens the recovery seal, proves to the server that it holds the vault key and joins the reader's devices.

Adding a device

Every new phone, tablet or computer has to receive the vault key before it can open a library. How it receives that key is the most important moment in the design.

A SIMPLE APPROACH

The most direct way to add a device would be:

1. The new device signs in and sends its public key to the server.
2. The server passes the key to one of the reader's existing devices.
3. The existing device seals the vault key to that public key and returns it.

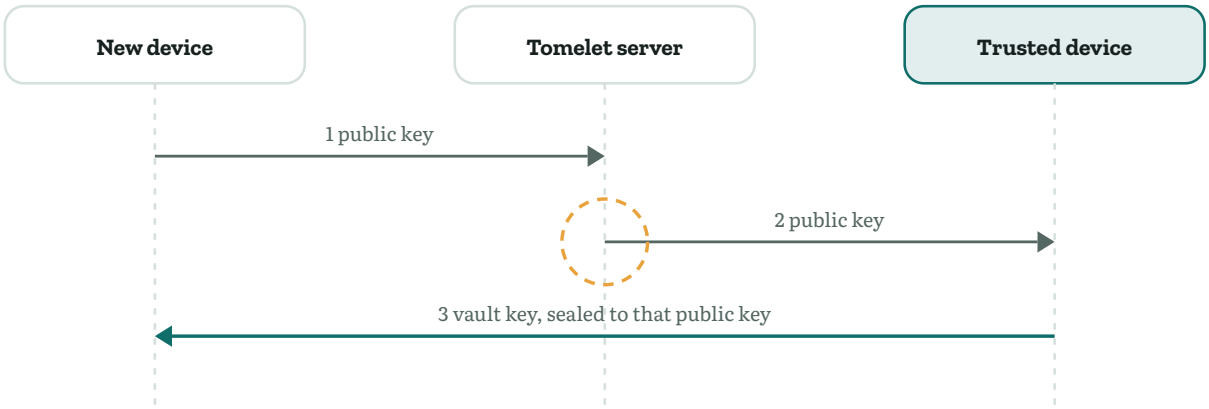


Figure 5. *In a simple design the server relays the new device's key, and the reader has to trust that relay.*

While this is straightforward, it trusts the server at the one moment it matters most. A compromised server could replace the new device's public key with its own and receive the vault key. That contradicts our zero-knowledge design tenet, so we added a verification step that the server cannot influence.

Adding a device with a verification code

1. The new device signs in and asks to join. Signing in alone gives it no access to the library.
2. The reader's trusted devices are notified. The new device and the approving device each show a 16-character code in four groups, derived from the SHA-256 hash of the new device's X25519 public key (80 bits).
3. The approving device computes its code itself, from the exact key it is about to seal the vault key to. It never displays a code supplied by the server.
4. When the reader sees that the codes match and approves, the approving device checks the new device's signature, seals the vault key to it and signs the approval with the vault authorization key. It refuses to seal to any key that does not produce the code it displayed.
5. The server verifies that signature before recording the approval, and the new device opens its seal.

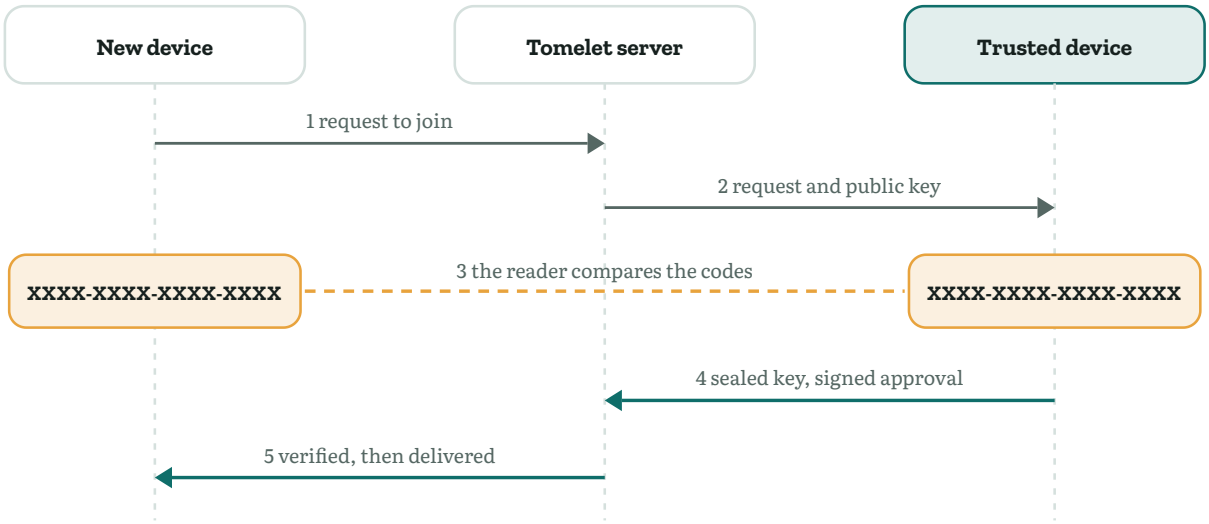


Figure 6. *The trusted device derives its code from the key it is about to seal to, so the reader's comparison confirms exactly which device receives the vault key.*

A request to join expires after seven days.

> **WHY COMPARE THE CODE?**
>
> What would an attack look like? Imagine a compromised server that intercepts a request to join and swaps in a public key of its own. The approving device derives its code from the key it has been given, so it shows a code made from the attacker's key, while the genuine new device shows a code made from its own.
>
> The reader sees two different codes, declines the request, and the vault key never reaches the attacker. Comparing the code takes a few seconds and closes the one gap a server could otherwise use.

Removing a device

A reader can remove a device from any of their other devices or from their account settings. The removed device is marked as such, its sessions end and the change is recorded in the account's security log.

Every sync request is checked against the device's current state, so a removed device is refused immediately, and it can only return as a new device that the reader approves again.

Encrypting and protecting your library

CHANGES

Every change a reader makes, whether a book added, a note written, a highlight moved, a page turned or a shelf renamed, is recorded as an encrypted change. What changed, which item it belongs to and when it happened are all inside the ciphertext, so the server cannot tell a highlight from a reading position, or one book from another.

FILES

Book files, covers and drawings of pen ink are stored as encrypted containers, each named by the SHA-256 hash of its ciphertext. The visible header of a container carries only what decrypting it in chunks requires: a format version, an identifier, a nonce prefix, the chunk size and the number of chunks. Its type and the hash of its contents stay inside the encryption.

INTEGRITY

- **Bound to its place:** Each change's authenticated data binds it to the account, the library, its own identity, the device that made it, its position in that device's sequence and the key it was made with, so it cannot be replayed elsewhere or under another identity.
- **Ordered and chained:** The server recognizes an exact duplicate, refuses a different change in a position already taken, and chains the library's history of changes with a running hash.
- **Signed requests:** Every sync and file request carries the device's Ed25519 signature over the account, the library, the device, its session, the request and a hash of its body. The server rebuilds and checks that statement exactly.
- **Files match their names:** The server accepts an upload only if its hash equals the name it was promised, and the reader's device checks every file it downloads against the record of it in the encrypted library.
- **One answer in two languages:** The apps and the server share known-answer test vectors, so the Dart and TypeScript implementations must agree byte for byte.

What our servers see

To sign a reader in, keep their changes in order and measure storage, the sync service holds the following. The Privacy Policy at tomelet.app/privacy describes the same data and why it is kept.

DATA	WHAT THE SERVICE HOLDS
Account	An identifier, the e-mail address and display name the sign-in provider shares, the join date, subscription status and storage used.
Sign-in	The providers in use and their account identifiers, when each was linked and last used, and hashed session tokens with their expiry.
Devices	An identifier, the platform, public keys, push token, app version and when the device was added and last seen. Device names are not stored.
Security log	Sign-ins, newly linked sign-in methods, sign-outs, device removals and account-deletion requests, with IP address and device, kept for up to 12 months.
Library	Encrypted changes and files, with their sizes, number, arrival times and the device that sent them. Titles, authors, notes, highlights, ink and positions stay encrypted.
Keys	Sealed copies of the vault key, and public keys.
Purchases	The store, product, status and dates of the subscription, as the subscription provider reports them. Card details never reach Tomelet.

Sizes, counts and timing are visible because the service needs them to store, order and measure a library, and together they show when a reader syncs and roughly how much they add. Features outside sync, such as looking up a book's details online, are described in the Privacy Policy along with the services they use.

Sign-in and sessions

Signing in and unlocking are deliberately separate. Signing in proves who a reader is; only an approved device or the Recovery Key can open their library.

- **Providers:** Readers sign in with Google, Apple, Facebook or a code sent by e-mail. Tomelet never sees their password for those providers.
- **Tomelet's own tokens:** Once a provider confirms who a reader is, Tomelet issues its own access token, a JSON Web Token signed with RS256 by a 2048-bit RSA key that never leaves AWS Key Management Service. It lasts one hour.
- **Refresh tokens:** 32 random bytes, stored by the server only as a SHA-256 hash. They last 90 days (30 in a web browser) and are replaced every time they are used. If an old token is ever presented again, its whole chain is revoked and the event is logged.
- **E-mail codes:** Six digits, valid for ten minutes, for one use and at most five attempts. A request for a code receives the same answer whether or not the address has an account.
- **One account per address:** Sign-ins that prove the same e-mail address reach the same account.
- **Signing out:** Ends that device's session and deletes its copy of the vault key. Signing out everywhere revokes every device's session.

Keys on your device

Each device keeps its own keys, its copy of the vault key and its session in the platform's secure storage.

PLATFORM	SECURE STORAGE
iPhone and iPad	The Keychain, readable only on this device while it is unlocked and never carried to a new device by backups.
Mac	The Keychain.
Android	The system's keystore-backed secure storage, excluded from Android backups.
Windows	Windows Data Protection, tied to the reader's Windows account.
Linux	The desktop keyring, or, where there is none, an encrypted file tied to the machine and readable only by the reader's user account.
Web browser	The browser's storage, sealed under a key the browser will not export.

The library on the device, its book files and reading database, is protected by the operating system's app sandbox and the device's own encryption.

Conclusion

Tomelet Sync gives readers one library across every device while keeping its contents private to them.

Books, their details, highlights, notes, pen ink and reading positions are encrypted on the reader's device before they leave it. The vault key that protects them exists only on the reader's approved devices and under their own Recovery Key, and a new device receives it only when the reader approves it by comparing a verification code on both screens, or enters their Recovery Key. Every change is authenticated and every sync request is signed, so our servers can store and order a library without being able to read or alter it.

IN SUMMARY

- **End-to-end encryption:** a library is encrypted with AES-256-GCM on the reader's device, and Tomelet stores only ciphertext.
- **Device-held keys:** the vault key lives on approved devices and, sealed, under the reader's Recovery Key.
- **Verified device approval:** the reader compares a code that the approving device derives from the key it seals to.
- **A Recovery Key the reader holds:** 128 random bits, shown once, restoring a library with no involvement from Tomelet.
- **Sign-in separate from unlocking:** a sign-in proves who a reader is; only an approved device or the Recovery Key opens a library.
- **Tamper-evident sync:** authenticated changes, signed sync requests and files that must match their names.

We publish this design so that anyone can examine it, and we welcome findings from the security community at security@tomelet.app under our vulnerability disclosure policy at tomelet.app/trust/vulnerability-disclosure. How Tomelet is built and operated is described in the companion engineering and operations white paper.

Tomelet

Read the book the way it was printed.

TRUST CENTER
tomelet.app/trust

SECURITY
security@tomelet.app

PRIVACY
privacy@tomelet.app