AirPlay also uses the authentication IC to verify that receivers have been approved by Apple. AirPlay audio and CarPlay video streams use the MFi-SAP (Secure Association Protocol), which encrypts communication between the accessory and device using AES128 in counter (CTR) mode. Ephemeral keys are exchanged using ECDH key exchange (Curve25519) and signed using the authentication IC's 1024-bit RSA key as part of the Station-to-Station (STS) protocol.

App security in macOS

App security overview in macOS

App security in macOS consists of a number of overlapping layers—the first of which is the option to run only signed and trusted apps from the App Store. In addition, macOS layers protections to ensure that apps downloaded from the internet are free of known malware. macOS offers technologies to detect and remove malware, and offers additional protections designed to prevent untrusted apps from accessing user data. Services from Apple such as Notarization and XProtect and MRT updates are designed to prevent malware installation and, when necessary, to provide for a quick and efficient detect-and-respond process to block and remove any malware that may have at first avoided detection. Ultimately, macOS users are free to operate within the security model that makes sense for them—including running completely unsigned and untrusted code.

App code signing process in macOS

All apps from the App Store are signed by Apple to ensure that they haven't been tampered with or altered. Apple signs any apps provided with Apple devices.

In macOS 10.15, all apps distributed outside the App Store must be signed by the developer using an Apple-issued Developer ID certificate (combined with a private key) and notarized by Apple to run under the default Gatekeeper settings. Apps developed in-house should also be signed with an Apple-issued Developer ID so that users can validate their integrity.

In macOS, code signing and notarization work independently—and can be performed by different actors—for different goals. Code signing is performed by the developer using their Developer ID certificate (issued by Apple), and verification of this signature proves to the user that a developer's software hasn't been tampered with since the developer built and signed it. Notarization can be performed by anyone in the software distribution chain and proves that Apple has been provided a copy of the code to check for malware and no known malware was found. The output of Notarization is a ticket, which is stored on Apple servers and can be optionally stapled to the app (by anyone) without invalidating the signature of the developer.

Mandatory Access Controls (MACs) require code signing to enable entitlements protected by the system. For example, apps requiring access through the firewall must be code signed with the appropriate MAC entitlement.

Gatekeeper and runtime protection in macOS

Gatekeeper

macOS includes a technology called Gatekeeper, which ensures that, by default, only trusted software runs on a user's Mac. When a user downloads and opens an app, a plug-in, or an installer package from outside the App Store, Gatekeeper verifies that the software is from an identified developer, is notarized by Apple to be free of known malicious content, and hasn't been altered. Gatekeeper also requests user approval before opening downloaded software for the first time to make sure the user hasn't been tricked into running executable code they believed to simply be a data file.

By default, Gatekeeper ensures that all downloaded software has been signed by the App Store or signed by a registered developer and notarized by Apple. Both the App Store review process and the notarization pipeline ensure that apps contain no known malware. Therefore, by default *all software in macOS is checked for known malicious content the first time it's opened, regardless of how it arrived on the Mac*.

Users and organizations have the option to allow only software installed from the App Store. Alternatively, users can override Gatekeeper policies to open any software unless restricted by a mobile device management (MDM) solution. Organizations can use MDM to configure Gatekeeper settings, including allowing software signed with alternate identities. Gatekeeper can also be completely disabled, if necessary.

Gatekeeper also protects against the distribution of malicious plug-ins with benign apps. Here, using the app triggers loading a malicious plug-in without the user's knowledge. When necessary, Gatekeeper opens apps from randomized, read-only locations, preventing the automatic loading of plug-ins distributed alongside the app.

Runtime protection

System files, resources, and the kernel are shielded from a user's app space. All apps from the App Store are sandboxed to restrict access to data stored by other apps. If an app from the App Store needs to access data from another app, it can do so only by using the APIs and services provided by macOS.

Protecting against malware in macOS

Apple operates a threat intelligence process to quickly identify and block malware. Malware defenses are structured in three layers:

1. *Prevent launch or execution of malware:* App Store or Gatekeeper and Notarization
2. *Block malware from running on customer systems:* Gatekeeper, Notarization, and XProtect
3. *Remediate malware that has executed:* MRT

The first layer of defense is designed to inhibit the distribution of malware, and prevent it from launching even once—this is the goal of the *App Store*, and *Gatekeeper* combined with *Notarization*.

The next layer of defense is to ensure that if malware appears on any Mac, it's quickly identified and blocked, both to halt spread and to remediate the Mac systems it's already gained a foothold on. *XProtect* adds to this defense, along with Gatekeeper and Notarization.

Finally, *MRT* acts to remediate malware that has managed to successfully execute.

These protections combine to support best-practice protection from viruses and malware. There are additional protections, particularly on a Mac with Apple silicon, to limit the potential damage of malware that does manage to execute. See Protecting app access to user data for ways that macOS can help protect user data from malware, and Operating system integrity for ways macOS can limit the actions malware can take on the system.

Notarization

Notarization is a malware scanning service provided by Apple. Developers who want to distribute apps for macOS outside the App Store submit their apps for scanning as part of the distribution process. Apple scans this software for known malware and, if none is found, issues a Notarization ticket. Typically, developers staple this ticket to their app so Gatekeeper can verify and launch the app, even offline.

Apple can also issue a revocation ticket for apps known to be malicious—even if they've been previously notarized. macOS regularly checks for new revocation tickets so that Gatekeeper has the latest information and can block launch of such files. This process can very quickly block malicious apps because updates happen in the background much more frequently than even the background updates that push new XProtect signatures. In addition, this protection can be applied to both apps that have been previously and those that haven't.

XProtect

macOS includes built-in antivirus technology called *XProtect* for the signature-based detection of malware. The system uses YARA signatures, a tool used to conduct signature-based detection of malware, which Apple updates regularly. Apple monitors for new malware infections and strains, and updates signatures automatically—independent from system updates—to help defend a Mac from malware infections. XProtect automatically detects and blocks the execution of known malware. In macOS 10.15 or later, XProtect checks for known malicious content whenever:

- An app is first launched
- An app has been changed (in the file system)
- XProtect signatures are updated

When XProtect detects known malware, the software is blocked and the user is notified and given the option to move the software to the Trash.

Note: Notarization is effective against known files (or file hashes) and can be used on apps that have been previously launched. The signature-based rules of XProtect are more generic than a specific file hash, so it can find variants that Apple has not seen. XProtect operates with a slower update cycle than notarization and scans only apps that have been changed or at first launch.

Malware Removal Tool

Should malware make its way onto a Mac, macOS also includes technology to remediate infections. The *Malware Removal Tool (MRT)* is an engine in macOS that remediates infections based on updates automatically delivered from Apple (as part of automatic updates of system data files and security updates). MRT removes malware upon receiving updated information, and it continues to check for infections on restart and login. MRT doesn't automatically reboot the Mac.

Automatic security updates

Apple issues the updates for XProtect and MRT automatically based on the latest threat intelligence available. By default, macOS checks for these updates daily. Notarization updates are distributed using CloudKit sync and are much more frequent.

Response process

When new malware is discovered, a number of steps may be performed:

- Any associated Developer ID certificates are revoked.
- Notarization revocation tickets are issued for all files (apps and associated files).
- XProtect signatures are developed and released.
- MRT signatures are developed and released.
- These signatures are also applied retroactively to previously notarized software, and any new detections can result in one or more of the previous actions occurring.

Ultimately, a malware detection launches a series of steps over the next seconds, hours, and days that follow to propagate the best protections possible to Mac users.

Controlling app access to files in macOS

Apple believes that users should have full transparency, consent, and control over what apps are doing with their data. In macOS 10.15, this model is enforced by the system to ensure that all apps must obtain user consent before accessing files in Documents, Downloads, Desktop, iCloud Drive, or network volumes. In macOS 10.13 or later, apps that require access to the full storage device must be explicitly added in System Preferences. In addition, accessibility and automation capabilities require user permission to ensure they don't circumvent other protections. Depending on the access policy, users may be prompted or must change the setting in System Preferences > Security & Privacy > Privacy:

Item	User prompted by app	User must edit system privacy settings
Accessibility		✓
Full internal storage access		✓
Files and folders *Note:* Includes: Desktop, Documents, Downloads, network volumes, and removable volumes	✓	
Automation (Apple events)	✓	

Items in the user's Trash are protected from any apps that are using Full Disk Access; the user won't get prompted for app access. If the user wants apps to access the files, they must be moved from the Trash to another location.

A user who enables FileVault on a Mac is asked to provide valid credentials before continuing the boot process and gain access to specialized startup modes. Without valid login credentials or a recovery key, the entire volume remains encrypted and is protected from unauthorized access, even if the physical storage device is removed and connected to another computer.

To protect data in an enterprise setting, IT should define and enforce FileVault configuration policies using mobile device management (MDM). Organizations have several options for managing encrypted volumes, including institutional recovery keys, personal recovery keys (that can optionally be stored with MDM for escrow), or a combination of both. Key rotation can also be set as a policy in MDM.

Secure features in the Notes app

Secure notes

The Notes app includes a secure notes feature that allows users to protect the contents of specific notes. Secure notes are end-to-end encrypted using a user-provided passphrase that is required to view the notes on iOS, iPadOS, macOS devices, and the iCloud website. Each iCloud account (including "On my" device accounts) can have a separate passphrase.

When a user secures a note, a 16-byte key is derived from the user's passphrase using PBKDF2 and SHA256. The note and all of its attachments are encrypted using AES with Galois/Counter Mode (AES-GCM). New records are created in Core Data and CloudKit to store the encrypted note, attachments, tag, and initialization vector. After the new records are created, the original unencrypted data is deleted. Attachments that support encryption include images, sketches, tables, maps, and websites. Notes containing other types of attachments can't be encrypted, and unsupported attachments can't be added to secure notes.

To view a secure note, the user must enter their passphrase or authenticate using Touch ID or Face ID. After successfully authenticating the user, whether to view or create a secure note, Notes opens a secure session. While the secure session is open, the user can view or secure other notes without additional authentication. However, the secure session applies only to notes protected with the provided passphrase. The user still needs to authenticate for notes protected by a different passphrase. The secure session is closed when:

- The user taps the Lock Now button in Notes
- Notes is switched to the background for more than 3 minutes (8 minutes in macOS)
- The iOS or iPadOS device locks

To change the passphrase on a secure note, the user must enter the current passphrase, because Touch ID and Face ID aren't available when changing the passphrase. After choosing a new passphrase, the Notes app rewraps, in the same account, the keys of all existing notes that are encrypted by the previous passphrase.

If a user mistypes the passphrase three times in a row, Notes shows a user-supplied hint if one was provided by the user at setup. If the user still doesn't remember their passphrase, they can reset it in Notes settings. This feature allows users to create new secure notes with a new passphrase, but it won't allow them to see previously secured notes. The previously secured notes can still be viewed if the old passphrase is remembered. Resetting the passphrase requires the user's iCloud account passphrase.

Shared notes

Notes that aren't end-to-end encrypted with a passphrase can be shared with others. Shared notes still use the CloudKit encrypted data type for any text or attachments that the user puts in a note. Assets are always encrypted with a key that's encrypted in the CKRecord. Metadata, such as the creation and modification dates, aren't encrypted. CloudKit manages the process by which participants can encrypt and decrypt each other's data.

Secure features in the Shortcuts app

In the Shortcuts app, shortcuts are optionally synced across Apple devices using iCloud. Shortcuts can also be shared with other users through iCloud. Shortcuts are stored locally in an encrypted format.

Custom shortcuts are versatile—they're similar to scripts or programs. When downloading shortcuts from the internet, the user is warned that the shortcut hasn't been reviewed by Apple and is given the opportunity to inspect the shortcut. To protect against malicious shortcuts, updated malware definitions are downloaded to identify malicious shortcuts at runtime.

Custom shortcuts can also run user-specified JavaScript on websites in Safari when invoked from the share sheet. To protect against malicious JavaScript that, for example, tricks the user into running a script on a social media website that harvests their data, the JavaScript is validated against the aforementioned malware definitions. The first time a user runs JavaScript on a domain, the user is prompted to allow shortcuts containing JavaScript to run on the current webpage for that domain.

Services security

Services security overview

Apple has built a robust set of services to help users get even more utility and productivity out of their devices. These services provide powerful capabilities for cloud storage, sync, password storage, authentication, payment, messaging, communications, and more, all while protecting users' privacy and the security of their data.

These services include iCloud, Sign in with Apple, Apple Pay, iMessage, Business Chat, FaceTime, Find My, and Continuity and may require an Apple ID or Managed Apple ID. In some cases, a Managed Apple ID can't be used with a specific service, like Apple Pay.

Note: Not all Apple services and content are available in all countries or regions.

Apple ID and Managed Apple ID

Apple ID security overview

Overview

An Apple ID is the account used to sign in to Apple services such as iCloud, iMessage, FaceTime, the iTunes Store, the App Store, the Apple TV app, Apple Books, and more. It's important for users to keep their Apple IDs secure to prevent unauthorized access to their accounts. To help with this, Apple IDs require strong passwords that:

- Must be at least eight characters in length
- Must contain both letters and numbers
- Must not contain more than three consecutive identical characters
- Can't be a commonly used password

Users are encouraged to exceed these guidelines by adding extra characters and punctuation marks to make their passwords even stronger.

Apple also notifies users in email and/or push notifications when important changes are made to their account—for example, if a password or billing information has been changed or the Apple ID has been used to sign in on a new device. If anything looks unfamiliar, users are instructed to change their Apple ID password immediately.

In addition, Apple employs a variety of policies and procedures designed to protect user accounts. These include limiting the number of retries for sign-in and password reset attempts, active fraud monitoring to help identify attacks as they occur, and regular policy reviews that allow Apple to adapt to any new information that could affect user security.

Note: The Managed Apple ID password policy is set by an administrator in Apple School Manager or Apple Business Manager.

Two-factor authentication

To help users further secure their accounts, Apple offers *two-factor authentication*—an extra layer of security for Apple IDs. It's designed to ensure that only the account's owner can access the account, even if someone else knows the password. With two-factor authentication, a user's account can be accessed on only trusted devices, such as the user's iPhone, iPad, iPod touch, or Mac, or on other devices after completing a verification from one of these trusted devices or a trusted phone number. To sign in for the first time on any new device, two pieces of information are required—the Apple ID password and a six-digit verification code that's displayed on the user's trusted devices or sent to a trusted phone number. By entering the code, the user confirms that they trust the new device and that it's safe to sign in. Because a password alone is no longer enough to access a user's account, two-factor authentication improves the security of the user's Apple ID and all the personal information they store with Apple. It's integrated directly into iOS, iPadOS, macOS, tvOS, watchOS, and the authentication systems used by Apple websites.

When a user signs in to an Apple website using a web browser, a second factor request is sent to all trusted devices associated with the user's iCloud account, requesting approval of the web session. If the user is signing in to an Apple website from a browser on a trusted device, they see the verification code displayed locally on the device they're using. When the user enters the code on that device, the web session is approved.

Account recovery

If an Apple ID account password is forgotten, a user can reset it on a trusted device. If a trusted device isn't available and the password is known, a user can use a trusted phone number can be used to authenticate through SMS verification. In addition, to provide immediate recovery for an Apple ID, a previously used passcode can be used to reset in conjunction with SMS. If these options aren't possible, the account recovery process must be followed. For more information, see the Apple Support article Recover your Apple ID when you can't reset your password.

Managed Apple ID security

Managed Apple IDs function much like an Apple ID but are owned and controlled by enterprise or educational organizations. These organizations can reset passwords, limit purchasing and communications such as FaceTime and Messages, and set up role-based permissions for employees, staff members, teachers, and students.

For Managed Apple IDs, some services are disabled (for example, Apple Pay, iCloud Keychain , HomeKit, and Find My).

Inspecting Managed Apple IDs

Managed Apple IDs also support *inspection*, which allows organizations to comply with legal and privacy regulations. An Apple School Manager administrator, manager, or teacher can inspect specific Managed Apple ID accounts.

Inspectors can monitor only accounts that are below them in the organization's hierarchy. For example, teachers can monitor students, managers can inspect teachers and students, and administrators can inspect managers, teachers, and students.

When inspecting credentials are requested using Apple School Manager, a special account is issued that has access to only the Managed Apple ID for which inspecting was requested. The inspector can then read and modify the user's content stored in iCloud or in CloudKit-enabled apps. Every request for auditing access is logged in Apple School Manager. The logs show who the inspector was, the Managed Apple ID the inspector requested access to, the time of the request, and whether the inspection was performed.

Managed Apple IDs and personal devices

Managed Apple IDs can also be used with personally owned iOS and iPadOS devices and Mac computers. Students sign in to iCloud using the Managed Apple ID issued by the institution and an additional home-use password, which serves as the second factor of the Apple ID two-factor authentication process. While students are using a Managed Apple ID on a personal device, iCloud Keychain isn't available, and the institution might restrict other features such as FaceTime or Messages. Any iCloud documents created by students when they are signed in are subject to audit as described previously in this section.

iCloud

iCloud security overview

iCloud stores a user's contacts, calendars, photos, documents, and more and keeps the information up to date across all of their devices automatically. iCloud can also be used by third-party apps to store and sync documents as well as key values for app data as defined by the developer. Users set up iCloud by signing in with an Apple ID and choosing which services they would like to use. Certain iCloud features, iCloud Drive, and iCloud Backup can be disabled by IT administrators using mobile device management (MDM) configuration profiles. The service is agnostic about what is being stored and handles all file content the same way, as a collection of bytes.

Each file is broken into chunks and encrypted by iCloud using AES128 and a key derived from each chunk's contents, with the keys using SHA256. The keys and the file's metadata are stored by Apple in the user's iCloud account. The encrypted chunks of the file are stored, without any user-identifying information or the keys, using both Apple and third-party storage services—such as Amazon Web Services or Google Cloud Platform—but these partners don't have the keys to decrypt the user's data stored on their servers.

iCloud Drive security

iCloud Drive adds account-based keys to protect documents stored in iCloud. iCloud Drive chunks and encrypts file contents and stores the encrypted chunks using third-party services. However, the file content keys are wrapped by record keys stored with the iCloud Drive metadata. These record keys are in turn protected by the user's iCloud Drive service key, which is then stored with the user's iCloud account. Users get access to their iCloud documents' metadata by having authenticated with iCloud, but they must also possess the iCloud Drive service key to expose protected parts of iCloud Drive storage.

iCloud Drive backup

iCloud also backs up information—including device settings, app data, photos, and videos in the Camera Roll, and conversations in the Messages app—daily over Wi-Fi. iCloud secures the content by encrypting it when it's sent over the internet, storing it in an encrypted format and using secure tokens for authentication. iCloud Backup occurs only when the device is locked, connected to a power source, and has Wi-Fi access to the internet. Because of the encryption used in iOS and iPadOS, iCloud Backup is designed to keep data secure while allowing incremental, unattended backup and restoration to occur.

When files are created in Data Protection classes that aren't accessible when the device is locked, their per-file keys are encrypted, using the class keys from the iCloud Backup keybag and backing the files up to iCloud in their original, encrypted state. All files are encrypted during transport and, when stored, encrypted using account-based keys, as described in CloudKit.

The iCloud Backup keybag contains asymmetric (Curve25519) keys for Data Protection classes that aren't accessible when the device is locked. The backup set is stored in the user's iCloud account and consists of a copy of the user's files and the iCloud Backup keybag. The iCloud Backup keybag is protected by a random key, which is also stored with the backup set. (The user's iCloud password isn't used for encryption, so changing the iCloud password won't invalidate existing backups.)

While the user's keychain database is backed up to iCloud, it remains protected by a UID-tangled key. This allows the keychain to be restored only to the same device from which it originated, and it means no one else, including Apple, can read the user's keychain items.

On restore, the backed-up files, iCloud Backup keybag, and the key for the keybag are retrieved from the user's iCloud account. The iCloud Backup keybag is decrypted using its key, then the per-file keys in the keybag are used to decrypt the files in the backup set, which are written as new files to the file system, thus reencrypting them according to their Data Protection class.

Security of iCloud Backup

The following content is backed up using iCloud Backup:

- Records for purchased music, movies, TV shows, apps, and books. A user's iCloud Backup includes information about purchased content present on the user's device, but not the purchased content itself. When the user restores from an iCloud Backup, their purchased content is automatically downloaded from the iTunes Store, the App Store, the Apple TV app, or Apple Books. Some types of content aren't downloaded automatically in all countries or regions, and previous purchases may be unavailable if they have been refunded or are no longer available in the store. Full purchase history is associated with a user's Apple ID.
- Photos and videos on a user's devices. Note that if a user turns on iCloud Photos in iOS 8.1 or later, iPadOS 13.1 or later, or OS X 10.10.3 or later, their photos and videos are already stored in iCloud, so they aren't included in the user's iCloud Backup.
- Contacts, calendar events, reminders, and notes
- Device settings
- App data
- Home screen and app organization
- HomeKit configuration
- Medical ID data
- Visual Voicemail password (requires the SIM card that was in use during backup)
- iMessage, Business Chat, text (SMS), and MMS messages (requires the SIM card that was in use during backup)

When Messages in iCloud is enabled, iMessage, Business Chat, text (SMS), and MMS messages are removed from the user's existing iCloud Backup and are instead stored in an end-to-end encrypted CloudKit container for Messages. The user's iCloud Backup retains a key to that container. If the user later disables iCloud Backup, that container's key is rolled, the new key is stored only in iCloud Keychain (inaccessible to Apple and any third parties), and new data written to the container can't be decrypted with the old container key.

The key used to restore the messages in iCloud Backup is placed in two locations, iCloud Keychain and a backup in CloudKit. The backup in CloudKit is done if iCloud Backup is enabled and unconditionally restored whether the user restores an iCloud backup or not.

CloudKit end-to-end encryption

Many Apple services, listed in the Apple Support article iCloud security overview, use end-to-end encryption with a CloudKit service key protected by iCloud Keychain syncing. For these CloudKit containers, the key hierarchy is rooted in iCloud Keychain and therefore shares the security characteristics of iCloud Keychain—namely, the keys are available only on the user's trusted devices, and not to Apple or any third party. If access to iCloud Keychain data is lost, the data in CloudKit is reset; and if data is available from the trusted local device, it's uploaded again to CloudKit. For more information, see Escrow security for iCloud Keychain.

Messages in iCloud also uses CloudKit end-to-end encryption with a CloudKit service key protected by iCloud Keychain syncing. If the user has enabled iCloud Backup, the CloudKit service key used for the Messages in iCloud container is backed up to iCloud to allow the user to recover their messages even if they have lost access to iCloud Keychain and their trusted devices. This iCloud service key is rolled whenever the user turns off iCloud Backup.

Situation	User recovery options for CloudKit end-to-end encryption
Access to trusted device	Data recovery possible using a trusted device or iCloud Keychain recovery.
No trusted devices	Data recovery only possible using iCloud Keychain recovery.
iCloud Backup enabled and access to trusted device	Data recovery possible using iCloud Backup, access to a trusted device, or iCloud Keychain recovery.
iCloud Backup enabled and no access to trusted device	Data recovery possible using iCloud Backup or iCloud Keychain recovery.
iCloud Backup disabled and access to trusted device	Data recovery possible using a trusted device or iCloud Keychain recovery.
Backup disabled and no trusted devices	Data recovery only possible using iCloud Keychain recovery.

Passcode and password management

Password security overview

iOS, iPadOS, and macOS make it easy for users to authenticate to third-party apps and websites that use passwords. The best way to manage passwords is not to have to use one. Sign in with Apple lets users sign in to third-party apps and websites without having to create and manage an additional account or password while protecting the sign-in with their two-factor authentication for Apple ID. For sites that don't support Sign in with Apple, the Automatic Strong Password feature enable a user's devices to automatically create, sync, and enter unique strong passwords for sites and apps. In iOS and iPadOS, passwords are saved to a special Password AutoFill keychain that's user controlled and manageable by going to Settings > Passwords.

In macOS, saved passwords can be managed in Safari Passwords preferences. This sync system can also be used to sync passwords that are manually created by the user.

Sign in with Apple security

Sign in with Apple is a privacy-friendly alternative to other single sign-on systems. It provides the convenience and efficiency of one-tap sign-in while giving the user more transparency and control over their personal information.

Sign in with Apple allows users to set up an account and sign in to apps and websites using the Apple ID they already have, and it gives them more control over their personal information. Apps can only ask for the user's name and email address when setting up an account, and the user always has a choice: They can share their personal email address with an app or choose to keep their personal email private and use the new Apple private email relay service instead. This email relay service shares a unique, anonymized email address that forwards to the user's personal address so they can still receive useful communication from the developer while maintaining a degree of privacy and control over their personal information.

Sign in with Apple is built for security. Every Sign in with Apple user is required to have two-factor authentication enabled for their Apple ID. Two-factor authentication helps secure not only the user's Apple ID but also the accounts they establish with their apps. Furthermore, Apple has developed and integrated a privacy-friendly antifraud signal into Sign in with Apple. This signal gives developers confidence that the new users they acquire are real people and not bots or scripted accounts.

Automatic strong passwords

When iCloud Keychain is enabled, iOS, iPadOS, and macOS create strong, random, unique passwords when users sign up for or change their password on a website in Safari. In iOS and iPadOS, automatic strong password generation is also available in apps. Users must opt out of using strong passwords. Generated passwords are saved in the keychain and kept up to date across devices with iCloud Keychain, when it's enabled.

By default, passwords generated by iOS and iPadOS are 20 characters long. They contain one digit, one uppercase character, two hyphens, and 16 lowercase characters. These generated passwords are strong, containing 71 bits of entropy.

Passwords are generated based on heuristics that determine whether a password-field experience is for password creation. If the heuristic fails to recognize a context-specific password being used at password creation, app developers can set `UITextContentType.newPassword` on their text field and web developers can set `autocomplete= "new-password"` on their <input> elements.

To ensure that generated passwords are compatible with the relevant services, apps and websites can provide rules. Developers provide these rules using `UITextInputPasswordRules` or the `passwordrules` attribute on their <input> elements. Devices then generate the strongest password they can that fulfills these rules.

Password AutoFill security

Password AutoFill automatically fills credentials stored in the keychain. The iCloud Keychain password manager and Password AutoFill provide the following features:

- Filling in credentials in apps and websites
- Generating strong passwords
- Saving passwords in both apps and websites in Safari
- Sharing passwords securely to a users' contacts
- Providing passwords to a nearby Apple TV that's requesting credentials

Generating and saving passwords within apps, as well as providing passwords to Apple TV, are available only in iOS and iPadOS.

Password AutoFill in apps

iOS and iPadOS allow users to input saved user names and passwords into credential-related fields in apps, similar to the way Password AutoFill works in Safari. In iOS and iPadOS, users tap a key affordance in the software keyboard's QuickType bar. In macOS, for apps built with Mac Catalyst, a Passwords drop-down menu appears below credential-related fields.

When an app is strongly associated with a website that uses the same app-website association mechanism and that's powered by the same apple-app-site-association file, the iOS and iPadOS QuickType bar and macOS drop-down menu directly suggest credentials for the app, if any are saved to the Password AutoFill Keychain. This allows users to choose to disclose Safari-saved credentials to apps with the same security properties, without those apps having to adopt an API.

Password AutoFill exposes no credential information to an app until a user consents to release a credential to the app. The credential lists are drawn from or presented out of the app's process.

When an app and website have a trusted relationship and a user submits credentials within an app, iOS and iPadOS may prompt the user to save those credentials to the Password AutoFill keychain for later use.

App access to saved passwords

iOS, iPadOS, and macOS apps can request the Password AutoFill keychain's help with signing a user in using ASAuthorizationPasswordProvider and SecAddSharedWebCredential. The password provider and its request can be used in conjunction with Sign in with Apple, so that the same API is called to help users sign into an app, regardless of whether the user's account is password based or was created using Sign in with Apple.

Apps can access saved passwords only if the app developer and website administrator have given their approval and the user has given consent. App developers express their intent to access Safari saved passwords by including an entitlement in their app. The entitlement lists the fully qualified domain names of associated websites, and the websites must place a file on their server listing the unique app identifiers of apps approved by Apple.

When an app with the com.apple.developer.associated-domains entitlement is installed, iOS and iPadOS make a TLS request to each listed website, requesting one of the following files:

- apple-app-site-association
- .well-known/apple-app-site-association

If the file lists the app identifier of the app being installed, then iOS and iPadOS mark the website and app as having a trusted relationship. Only with a trusted relationship will calls to these two APIs result in a prompt to the user, who must agree before any passwords are released to the app, updated, or deleted.

Password security recommendations

Overview

The Password AutoFill passwords list in iOS, iPadOS, and macOS indicates which of a user's saved passwords will be reused with other websites, passwords that are considered weak, and passwords that have been compromised by a data leak.

Using the same password for more than one service may leave those accounts vulnerable to a credential-stuffing attack. If a service is breached and passwords are leaked, attackers may try the same credentials on other services to compromise additional accounts. Passwords are marked as *reused* if the same password is seen used for more than one saved password across different domains.

Passwords are marked *weak* if they may be easily guessed by an attacker. iOS, iPadOS, and macOS detect common patterns used to create memorable passwords, such as using words found in a dictionary, common character substitutions (such as using "p4ssw0rd" instead of "password"), patterns found on a keyboard (such as "q12we34r" from a QWERTY keyboard), or repeated sequences (such as "123123"). These patterns are often used to create passwords that satisfy minimum password requirements for services, but are also commonly used by attackers attempting to obtain a password using brute force.

Because many services specifically require a four- or six-digit PIN code, these short passcodes are evaluated with different rules. PIN codes are considered weak if they are one of the most common PIN codes, if they are an increasing or decreasing sequence such as "1234" or "8765," or if they follow a repetition pattern, such as "123123" or "123321."

Passwords are marked *leaked* if the Password Monitoring feature can claim they have been present in a data leak. For more information, see Password Monitoring.

Weak, reused, and leaked passwords are either indicated in the list of passwords (macOS) or present in the dedicated Security Recommendations interface (iOS and iPadOS). If the user logs in to a website in Safari using a previously saved password that's very weak or that's been compromised by a data leak, they're shown an alert strongly encouraging them to upgrade to an automatic strong password.

Upgrading account authentication security with Account Authentication Modification Extensions

Apps that implement an Account Authentication Modification Extension can provide easy, tap-of-a-button upgrades for password-based accounts—namely, they can switch to using Sign in with Apple or an automatic strong password. This extension point is available in iOS and iPadOS.

If an app has implemented the extension point and is installed on device, users see extension upgrade options when viewing Security Recommendations for credentials associated with the app in the iCloud Keychain password manager in Settings. The upgrades are also offered when users sign in to the app with the at-risk credential. Apps have the ability to tell the system not to prompt users with upgrade options after signing in. Using new AuthenticationServices API, apps can also invoke their extensions and perform upgrades themselves, ideally from an account settings or account management screen in the app.

Apps can choose to support strong password upgrades, Sign in with Apple upgrades, or both. In a strong password upgrade, the system generates an automatic strong password for the user. If necessary, the app can provide custom password rules to follow when generating the new password. When a user switches an account from using a password to using Sign in with Apple, the system provides a new Sign in with Apple credential to the extension to associate the account with. The user's Apple ID email isn't provided as part of the credential. After a successful Sign in with Apple upgrade, the system deletes the previously used password credential from the user's keychain if it's saved there.

Account Authentication Modification Extensions have the opportunity to perform additional user authentication before performing an upgrade. For upgrades initiated within the password manager or after signing in to an app, the extension provides the user name and password for the account to upgrade. For in-app upgrades, only the user name is provided. If the extension requires further user authentication, it can request to show a custom user interface before moving on with the upgrade. The intended use case for showing this user interface is to have the user enter a second factor of authentication to authorize the upgrade.

Password Monitoring

Password Monitoring is a feature that matches passwords stored in your Password AutoFill keychain against a continuously updated and curated list of passwords known to have been exposed in leaks from different online organizations. If the feature is turned on, the monitoring protocol continuously matches your Password AutoFill keychain passwords against the curated list.

How monitoring works

The user's device continuously performs round-robin checks on a user's passwords, querying on an interval that's independent of the user's passwords, ensuring that verification states remain up to date with the current curated list of leaked passwords. To prevent leakage of information related to how many unique passwords a user has, requests are batched and performed in parallel. A fixed number of passwords are verified in parallel on each check, and should the user have fewer than this number, random passwords are generated and added to the queries to make up the difference.

How matching is done

Matching is done in a two-part process. The most commonly leaked passwords are contained within a local list on the user's device. If the user's password occurs on this list, the user is immediately notified without any external interaction. This ensures that no information is leaked about the passwords a user has that are most at risk due to a password breach.

If the password isn't contained on the most frequent list, then it is matched against less frequently leaked passwords.

Comparing users' passwords against a curated list

To verify whether a password not present in the local list is a match involves some interaction with Apple servers. To ensure that legitimate users' passwords aren't sent to Apple, a form of cryptographic *private set intersection* is deployed that compares the users' passwords against a large set of leaked passwords. This ensures that for passwords less at risk of breach, little information is shared with Apple. For a user's password, this information is limited to a 15-bit prefix of a cryptographic hash. The removal of the most frequently leaked passwords from this interactive process, using the local list of most commonly leaked passwords, reduces the delta in relative frequency of passwords in the web services buckets, making it impractical to infer user passwords from these lookups.

The underlying protocol partitions the list of curated passwords, which contained approximately 1.5 billion passwords at the time of this writing, into 2^{15} different buckets. The bucket a password belongs to is based on the first 15 bits of the SHA256 hash value of the password. Additionally, each leaked password, pw, is associated with an elliptic curve point on the NIST P256 curve: $P_{pw} = \alpha \cdot H_{SWU}(pw)$, where α is a secret random key known only to Apple and H_{SWU} is a random oracle function that maps passwords to curve points based on the Shallue-van de Woestijne-Ulas method. This transformation is designed to computationally hide the values of passwords and prevents revealing newly leaked passwords through Password Monitoring.

To compute the private set intersection, the user's device determines the bucket the user's password belongs to using λ, the 15-bit prefix of SHA256(upw), where upw is one of the user's passwords. The device generates their own random constant, β, and sends the point $P_c = \beta \cdot H_{SWU}(upw)$ to the server, along with a request for the bucket corresponding to λ. Here β hides information about the user's password and limits to λ the information exposed from the password to Apple. Finally, the server takes the point sent by the user's device, computes $\alpha P_c = \alpha\beta \cdot H_{SWU}(upw)$, and returns it, along with the appropriate bucket of points—$B\lambda = \{ P_{pw} \mid \text{SHA256(pw) begins with prefix } \lambda\}$—to the device.

The returned information allows the device to compute $B'_\lambda = \{\beta \cdot P_{pw} \mid P_{pw} \in B_\lambda\}$, and ascertains that the user's password has been leaked if $\alpha P_c \in B'_\lambda$.

Sending passwords to other users or Apple devices

Saving credentials to another device with AirDrop

When iCloud is enabled, users can AirDrop a saved credential—including the websites it's saved for, its user name, and its password—to another device. Sending credentials with AirDrop always operates in Contacts Only mode, regardless of the user's settings. On the receiving device, after user consent, the credential are stored in the user's Password AutoFill Keychain.

Filling in credentials in apps on Apple TV

Password AutoFill is available to fill credentials in apps on Apple TV. When the user focuses on a user name or password text field in tvOS, Apple TV begins advertising a request for Password AutoFill over Bluetooth Low Energy (BLE).

Any nearby iPhone, iPad, or iPod touch displays a prompt inviting the user to share a credential with Apple TV. Here's how the encryption method is established:

- If the device and Apple TV uses the same iCloud account, encryption between the devices happens automatically.
- If the device is signed in to an iCloud account other than the one used by Apple TV, the user is prompted to establish an encrypted connection through use of a PIN code. To receive this prompt, iPhone must be unlocked and in close proximity to the Siri Remote paired to that Apple TV.

After the encrypted connection is made using BLE link encryption, the credential is sent to Apple TV and is automatically filled in to the relevant text fields on the app.

Credential provider extensions

In iOS, iPadOS, and macOS, users can designate a participating third-party app as a credential provider for Password AutoFill in Passwords settings (iOS and iPadOS) or in Extensions settings in System Preferences (macOS). This mechanism is built on app extensions. The credential provider extension must provide a view for choosing credentials, and the extension can optionally provide metadata about saved credentials so they can be offered directly on the QuickType bar (iOS and iPadOS) or in an autocomplete suggestion (macOS). The metadata includes the website of the credential and the associated user name but not its password. iOS, iPadOS, and macOS communicate with the extension to get the password when the user chooses to fill a credential into an app or a website in Safari. Credential metadata is stored inside the credential provider app's container and is automatically removed when an app is uninstalled.

iCloud Keychain

iCloud Keychain security overview

iCloud allows users to securely sync their passwords between iOS and iPadOS devices and Mac computers without exposing that information to Apple. In addition to strong privacy and security, other goals that heavily influenced the design and architecture of iCloud Keychain were ease of use and the ability to recover a keychain. iCloud Keychain consists of two services: keychain syncing and keychain recovery.

Apple designed iCloud Keychain and keychain recovery so that a user's passwords are still protected under the following conditions:

- A user's iCloud account is compromised.
- iCloud is compromised by an external attacker or employee.
- A third party accesses user accounts.

Password manager integration with iCloud Keychain

iOS, iPadOS, and macOS can automatically generate cryptographically strong random strings to use as account passwords in Safari. iOS and iPadOS can also generate strong passwords for apps. Generated passwords are stored in the keychain and synced to other devices. Keychain items are transferred from device to device, traveling through Apple servers, but are encrypted in such a way that Apple and other devices can't read their contents.

Secure keychain syncing

When a user enables iCloud Keychain for the first time, the device establishes a circle of trust and creates a syncing identity for itself. The syncing identity consists of a private key and a public key. The public key of the syncing identity is put in the circle, and the circle is signed twice: first by the private key of the syncing identity, then again with an asymmetric elliptical key (using P-256) derived from the user's iCloud account password. Also stored with the circle are the parameters (random salt and iterations) used to create the key that is based on the user's iCloud password.

Syncing circle is placed in user's iCloud

The signed syncing circle is placed in the user's iCloud key-value storage area. It can't be read without knowing the user's iCloud password and can't be modified validly without having the private key of the syncing identity of its member.

When the user turns on iCloud Keychain on another device, iCloud Keychain notices that the user has a previously established syncing circle in iCloud that it isn't a member of. The device creates its syncing identity key pair, then creates an application ticket to request membership in the circle. The ticket consists of the device's public key of its syncing identity, and the user is asked to authenticate with their iCloud password. The elliptical key-generation parameters are retrieved from iCloud and generate a key that's used to sign the application ticket. Finally, the application ticket is placed in iCloud.

How a user's other devices are added to the syncing circle

When the first device sees that an application ticket has arrived, it asks for the user to acknowledge that a new device is asking to join the syncing circle. The user enters their iCloud password, and the application ticket is verified as signed by a matching private key. Now, the users who generated the request to join the circle can join it.

Upon the user's approval to add the new device to the circle, the first device adds the public key of the new member to the syncing circle and signs it again with both its syncing identity and the key derived from the user's iCloud password. The new syncing circle is placed in iCloud, where it's similarly signed by the new member of the circle.

There are now two members of the signing circle, and each member has the public key of its peer. They now begin to exchange individual keychain items through iCloud key-value storage or store them in CloudKit, whichever is most appropriate for the situation. If both circle members have the same item, the one with the most recent modification date is synced. If the other member has the item and the modification dates are identical, items are skipped. Each item that's synced is encrypted so that it can be decrypted only by a device within the user's circle of trust; it can't be decrypted by any other devices or by Apple.

This process is repeated as new devices join the syncing circle. For example, when a third device joins, the confirmation appears on both of the other user's devices. The user can approve the new member from either of those devices. As new peers are added, each peer syncs with the new one to ensure that all members have the same keychain items.

Only certain items are synced

However, the entire keychain isn't synced. Some items are device specific, such as VPN identities, and shouldn't leave the device. Only items with the `kSecAttrSynchronizable` attribute are synced. Apple has set this attribute for Safari user data (including user names, passwords, and credit card numbers) as well as for Wi-Fi passwords and HomeKit encryption keys.

Additionally, by default, keychain items added by third-party apps don't sync. Developers must set the `kSecAttrSynchronizable` attribute when adding items to the keychain.

Secure iCloud Keychain recovery

iCloud Keychain recovery provides a way for users to optionally escrow their keychain with Apple without allowing Apple to read the passwords and other data it contains. Even if the user has only a single device, keychain recovery provides a safety net against data loss. This is particularly important when Safari is used to generate random, strong passwords for web accounts, because the only record of those passwords is in the keychain.

A cornerstone of keychain recovery is secondary authentication and a secure escrow service, created by Apple specifically to support this feature. The user's keychain is encrypted using a strong passcode, and the escrow service provides a copy of the keychain only if a strict set of conditions are met.

Use of secondary authentication

There are several ways to establish a strong passcode:

- If two-factor authentication is enabled for the user's account, the device passcode is used to recover an escrowed keychain.
- If two-factor authentication isn't set up, the user is asked to create an iCloud security code by providing a six-digit passcode. Alternatively, without two-factor authentication, users can specify their own, longer code, or they can let their devices create a cryptographically random code that they can record and keep on their own.

Keychain escrow process

After the passcode is established, the keychain is escrowed with Apple. The iOS, iPadOS, or macOS device first exports a copy of the user's keychain and then encrypts it wrapped with keys in an asymmetric keybag and places it in the user's iCloud key-value storage area. The keybag is wrapped with the user's iCloud security code and with the public key of the hardware security module (HSM) cluster that stores the escrow record. This becomes the user's iCloud escrow record. For HSA2 accounts, the keychain is also stored in CloudKit and wrapped to intermediate keys that are recoverable only with the contents of the iCloud escrow record, thereby providing the same level of protection.

Note: If the user decides to accept a cryptographically random security code instead of specifying their own or using a four-digit value, no escrow record is necessary. Instead, the iCloud security code is used to wrap the random key directly.

Besides establishing a security code, users must register a phone number. This provides a secondary level of authentication during keychain recovery. The user receives an SMS message that must be replied to for the recovery to proceed.

Escrow security for iCloud Keychain

iCloud provides a secure infrastructure for keychain escrow to ensure that only authorized users and devices can perform a recovery. Topographically positioned behind iCloud are clusters of hardware security modules (HSMs) that guard the escrow records. As described previously, each has a key that is used to encrypt the escrow records under their watch.

To recover a keychain, users must authenticate with their iCloud account and password and respond to an SMS sent to their registered phone number. After this is done, users must enter their iCloud security code. The HSM cluster verifies that a user knows their iCloud security code using the Secure Remote Password (SRP) protocol; the code itself isn't sent to Apple. Each member of the cluster independently verifies that the user hasn't exceeded the maximum number of attempts allowed to retrieve their record, as discussed below. If a majority agree, the cluster unwraps the escrow record and sends it to the user's device.

Next, the device uses the iCloud security code to unwrap the random keys used to encrypt the user's keychain. With that key, the keychain—retrieved from iCloud key-value storage and CloudKit—is decrypted and restored onto the device. iOS, iPadOS, and macOS allow only 10 attempts to authenticate and retrieve an escrow record. After several failed attempts, the record is locked and the user must call Apple Support to be granted more attempts. After the 10th failed attempt, the HSM cluster destroys the escrow record and the keychain is lost forever. This provides protection against a brute-force attempt to retrieve the record, at the expense of sacrificing the keychain data in response.

These policies are coded in the HSM firmware. The administrative access cards that permit the firmware to be changed have been destroyed. Any attempt to alter the firmware or access the private key causes the HSM cluster to delete the private key. Should this occur, the owner of each keychain protected by the cluster receives a message informing them that their escrow record has been lost. They can then choose to reenroll.

Apple Pay

Apple Pay security overview

With Apple Pay, users can use supported iPhone, iPad, Mac, and Apple Watch devices to pay in an easy, secure, and private way in stores, apps, and on the web in Safari. Users can also add Apple Pay–enabled transit and student ID cards to Apple Wallet. It's simple for users, and it's built with integrated security in both hardware and software.

Apple Pay is also designed to protect the user's personal information. Apple Pay doesn't collect any transaction information that can be tied back to the user. Payment transactions are between the user, the merchant, and the card issuer.

Apple Pay component security

Secure Element

The Secure Element is an industry-standard, certified chip running the Java Card platform, which is compliant with financial industry requirements for electronic payments. The Secure Element IC and the Java Card platform are certified in accordance with the EMVCo Security Evaluation process. After the successful completion of the security evaluation, EMVCo issues unique IC and platform certificates.

The Secure Element IC has been certified based on the Common Criteria standard.

NFC controller

The NFC controller handles Near Field Communication protocols and routes communication between the Application Processor and the Secure Element, and between the Secure Element and the point-of-sale terminal.

Apple Wallet

Apple Wallet is used to add and manage credit, debit, and store cards and to make payments with Apple Pay. Users can view their cards and may be able to view additional information provided by their card issuer, such as their card issuer's privacy policy, recent transactions, and more in Apple Wallet. Users can also add cards to Apple Pay in:

- Setup Assistant and Settings for iOS and iPadOS
- The Watch app for Apple Watch
- Wallet & Apple Pay in System Preferences for Mac computers with Touch ID

In addition, Apple Wallet allows users to add and manage transit cards, rewards cards, boarding passes, tickets, gift cards, student ID cards, and more.

Secure Enclave

On iPhone, iPad, Apple Watch, and Mac computers with Touch ID, the Secure Enclave manages the authentication process and enables a payment transaction to proceed.

On Apple Watch, the device must be unlocked, and the user must double-click the side button. The double-click is detected and passed directly to the Secure Element or Secure Enclave, where available, without going through the Application Processor.

Apple Pay servers

The Apple Pay servers manage the setup and provisioning of credit, debit, transit, and student ID cards in the Wallet app. The servers also manage the Device Account Numbers stored in the Secure Element. They communicate both with the device and with the payment network or card issuer servers. The Apple Pay servers are also responsible for reencrypting payment credentials for payments within apps or on the web.

How Apple Pay uses the Secure Element and NFC controller

Secure Element

The Secure Element hosts a specially designed applet to manage Apple Pay. It also includes applets certified by payment networks or card issuers. Credit, debit, or prepaid card data is sent from the payment network or card issuer encrypted to these applets using keys that are known only to the payment network or card issuer and the applets' security domain. This data is stored within these applets and protected using the Secure Element's security features. During a transaction, the terminal communicates directly with the Secure Element through the near-field-communication (NFC) controller over a dedicated hardware bus.

NFC controller

As the gateway to the Secure Element, the NFC controller ensures that all contactless payment transactions are conducted using a point-of-sale terminal that is in close proximity with the device. Only payment requests arriving from an in-field terminal are marked by the NFC controller as contactless transactions.

After a credit, debit, or prepaid card (including store cards) payment is authorized by the cardholder using Touch ID, Face ID, or a passcode, or on an unlocked Apple Watch by double-clicking the side button, contactless responses prepared by the payment applets within the Secure Element are exclusively routed by the controller to the NFC field. Consequently, payment authorization details for contactless payment transactions are contained to the local NFC field and are never exposed to the Application Processor. In contrast, payment authorization details for payments within apps and on the web are routed to the Application Processor, but only after encryption by the Secure Element to the Apple Pay server.

Credit, debit, and prepaid cards

Card provisioning security overview

When a user adds a credit, debit, or prepaid card (including store cards) to Apple Wallet, Apple securely sends the card information, along with other information about user's account and device, to the card issuer or card issuer's authorized service provider. Using this information, the card issuer determines whether to approve adding the card to Apple Wallet.

As part of the card provisioning process, Apple Pay uses three server-side calls to send and receive communication with the card issuer or network: Required Fields, Check Card, and Link and Provision. The card issuer or network uses these calls to verify, approve, and add cards to Apple Wallet. These client-server sessions use TLS 1.2 to transfer the data.

Full card numbers aren't stored on the device or on Apple Pay servers. Instead, a unique Device Account Number is created, encrypted, and then stored in the Secure Element. This unique Device Account Number is encrypted in such a way that Apple can't access it. The Device Account Number is unique and different from most credit or debit card numbers; the card issuer or payment network can prevent its use on a magnetic stripe card, over the phone, or on websites. The Device Account Number in the Secure Element is never stored on Apple Pay servers or backed up to iCloud, and it is isolated from iOS, iPadOS, and watchOS devices and from Mac computers with Touch ID.

Cards for use with Apple Watch are provisioned for Apple Pay using the Apple Watch app on iPhone, or within a card issuer's iPhone app. Adding a card to Apple Watch requires that the watch be within Bluetooth communications range. Cards are specifically enrolled for use with Apple Watch and have their own Device Account Numbers, which are stored within the Secure Element on the Apple Watch.

When credit, debit, or prepaid cards (including store cards) are added, they appear in a list of cards during Setup Assistant on devices that are signed in to the same iCloud account. These cards remain in this list for as long as they are active on at least one device. Cards are removed from this list after they have been removed from all devices for 7 days. This feature requires two-factor authentication to be enabled on the respective iCloud account.

Adding credit or debit cards to Apple Pay

Adding credit or debit cards manually

To add a card manually, the name, card number, expiration date, and CVV are used to facilitate the provisioning process. From within Settings, the Wallet app, or the Apple Watch app, users can enter that information either by typing or by using the device's camera. When the camera captures the card information, Apple attempts to populate the name, card number, and expiration date. The photo is never saved to the device or stored in the photo library. After all the fields are filled in, the Check Card process verifies the fields other than the CVV. They are then encrypted and sent to the Apple Pay server.

If a terms and conditions ID is returned with the Check Card process, Apple downloads and displays the terms and conditions of the card issuer to the user. If the user accepts the terms and conditions, Apple sends the ID of the terms that were accepted as well as the CVV to the Link and Provision process. Additionally, as part of the Link and Provision process, Apple shares information from the device with the card issuer or network, like information about the user's iTunes and App Store account activity (for example, whether the user has a long history of transactions within iTunes), information about the user's device (for example, phone number, name, and model of the user's device plus any companion Apple device necessary to set up Apple Pay), and the user's approximate location at the time the user adds their card (if the user has Location Services enabled). Using this information, the card issuer determines whether to approve adding the card to Apple Pay.

As the result of the Link and Provision process, two things occur:

- The device begins to download the Wallet pass file representing the credit or debit card.
- The device begins to bind the card to the Secure Element.

The pass file contains URLs to download card art, metadata about the card such as contact information, the related issuer's app, and supported features. It also contains the pass state, which includes information such as whether the personalizing of the Secure Element has completed, whether the card is currently suspended by the card issuer, or whether additional verification is required before the card can make payments with Apple Pay.

Adding credit or debit cards from an iTunes Store account

For a credit or debit card on file with iTunes, the user may be required to reenter their Apple ID password. The card number is retrieved from iTunes, and the Check Card process is initiated. If the card is eligible for Apple Pay, the device downloads and displays terms and conditions, then send along the term's ID and the card security code to the Link and Provision process. Additional verification may occur for iTunes account cards on file.

Adding credit or debit cards from a card issuer's app

When the app is registered for use with Apple Pay, keys are established for the app and for the card issuer's server. These keys are used to encrypt the card information that's sent to the card issuer, which prevents the information from being read by the Apple device. The provisioning flow is similar to that used for manually added cards, described previously, except one-time passwords are used in lieu of the CVV.

Adding additional verification

A card issuer can decide whether a credit or debit card requires additional verification. Depending on what's offered by the card issuer, the user may be able to choose between different options for additional verification, such as a text message, email, customer service call, or a method in an approved third-party app to complete the verification. For text messages or email, the user selects from contact information the issuer has on file. A code is sent, which must be entered into the Wallet app, Settings, or the Apple Watch app. For customer service or verification using an app, the issuer performs their own communication process.

Payment authorization with Apple Pay

For devices having a Secure Enclave, a payment can be made only after it receives authorization from the Secure Enclave. On iPhone or iPad, this involves confirming that the user has authenticated with Touch ID, Face ID, or the device passcode. Touch ID or Face ID, if available, is the default method, but the passcode can be used at any time. A passcode is automatically offered after three unsuccessful attempts to match a fingerprint, or two unsuccessful attempts to match a face; after five unsuccessful attempts, the passcode is required. A passcode is also required when Touch ID or Face ID isn't configured or isn't enabled for Apple Pay. For a payment to be made on Apple Watch, the device must be unlocked with passcode and the side button must be double-clicked.

Using a shared pairing key

Communication between the Secure Enclave and the Secure Element takes place over a serial interface, with the Secure Element connected to the NFC controller, which in turn is connected to the Application Processor. Though not directly connected, the Secure Enclave and Secure Element can communicate securely using a shared pairing key that's provisioned during the manufacturing process. The encryption and authentication of the communication are based on AES, with cryptographic nonces used by both sides to protect against replay attacks. The pairing key is generated inside the Secure Enclave from its UID key and the Secure Element unique identifier. The pairing key is then securely transferred from the Secure Enclave to a hardware security module (HSM) in the factory, which has the key material required to then inject the pairing key into the Secure Element.

Authorizing a secure transaction

When the user authorizes a transaction, which includes a physical gesture communicated directly to the Secure Enclave, the Secure Enclave then sends signed data about the type of authentication and details about the type of transaction (contactless or within apps) to the Secure Element, tied to an Authorization Random (AR) value. The AR value is generated in the Secure Enclave when a user first provisions a credit card and persists while Apple Pay is enabled, protected by the Secure Enclave encryption and anti-rollback mechanism. It's securely delivered to the Secure Element by leveraging the pairing key. On receipt of a new AR value, the Secure Element marks any previously added cards as deleted.

Using a payment cryptogram for dynamic security

Payment transactions originating from the payment applets include a payment cryptogram along with a Device Account Number. This cryptogram, a one-time code, is computed using a transaction counter and a key. The transaction counter is incremented for each new transaction. The key is provisioned in the payment applet during personalization and is known by the payment network and/or the card issuer. Depending on the payment scheme, other data may also be used in the calculation, including:

- A Terminal Unpredictable Number, for near-field-communication (NFC) transactions
- An Apple Pay server nonce, for transactions within apps

These security codes are provided to the payment network and to the card issuer, which allows the issuer to verify each transaction. The length of these security codes may vary based on the type of transaction.

Paying with cards using Apple Pay

Paying with cards in stores

If iPhone or Apple Watch is on and detects an NFC field, it presents the user with the requested card (if automatic selection is turned on for that card) or the default card, which is managed in Settings. The user can also go to the Wallet app and choose a card, or when the device is locked, can:

- Double-click the Home button on devices with Touch ID
- Double-click the side button on devices with Face ID

Next, before information is transmitted, the user must authenticate using Touch ID, Face ID, or their passcode. When Apple Watch is unlocked, double-clicking the side button activates the default card for payment. No payment information is sent without user authentication.

After the user authenticates, the Device Account Number and a transaction-specific dynamic security code are used when processing the payment. Neither Apple nor a user's device sends the full actual credit or debit card numbers to merchants. Apple may receive anonymous transaction information such as the approximate time and location of the transaction, which helps improve Apple Pay and other Apple products and services.

Paying with cards within apps

Apple Pay can also be used to make payments on iPhone, iPad, Mac, and Apple Watch apps. When users pay within apps using Apple Pay, Apple receives the encrypted transaction information. Before that information is sent to the developer or merchant, Apple reencrypts the transaction with a developer-specific key. Apple Pay retains anonymous transaction information, such as approximate purchase amount. This information can't be tied to the user and never includes what the user is buying.

When an app initiates an Apple Pay payment transaction, the Apple Pay servers receive the encrypted transaction from the device prior to the merchant receiving it. The Apple Pay servers then reencrypt the transaction with a merchant-specific key before relaying it to the merchant.

When an app requests a payment, it calls an API to determine whether the device supports Apple Pay and whether the user has credit or debit cards that can make payments on a payment network accepted by the merchant. The app requests any pieces of information it needs to process and fulfill the transaction, such as the billing and shipping address, and contact information. The app then asks iOS, iPadOS, or watchOS to present the Apple Pay sheet, which requests information for the app as well as other necessary information, such as the card to use.

At this time, the app is presented with city, state, and zip code information to calculate the final shipping cost. The full set of requested information isn't provided to the app until the user authorizes the payment with Touch ID, Face ID, or the device passcode. After the payment is authorized, the information presented in the Apple Pay sheet is transferred to the merchant.

App payment authorization

When the user authorizes the payment, a call is made to the Apple Pay servers to obtain a cryptographic nonce, which is similar to the value returned by the NFC terminal used for in-store transactions. The nonce, along with other transaction data, is passed to the Secure Element to generate a payment credential that's encrypted with an Apple key. When the encrypted payment credential comes out of the Secure Element, it's passed to the Apple Pay servers, which decrypt the credential, verify the nonce in the credential against the nonce originally sent by the Apple Pay servers, and reencrypt the payment credential with the merchant key associated with the Merchant ID. The payment is then returned to the device, which hands it back to the app through the API. The app then passes it along to the merchant system for processing. The merchant can then decrypt the payment credential with its private key for processing. This, together with the signature from Apple's servers, allows the merchant to verify that the transaction was intended for this particular merchant.

The APIs require an entitlement that specifies the supported Merchant IDs. An app can also include additional data (such as an order number or customer identity) to send to the Secure Element to be signed, ensuring that the transaction can't be diverted to a different customer. This is accomplished by the app developer, who can specify applicationData on the PKPaymentRequest. A hash of this data is included in the encrypted payment data. The merchant is then responsible for verifying that their applicationData hash matches what's included in the payment data.

Paying with cards at websites

Apple Pay can be used to make payments at websites on iPhone, iPad, Apple Watch, and Mac computers with Touch ID. Apple Pay transactions can also start on a Mac and be completed on an Apple Pay–enabled iPhone or Apple Watch using the same iCloud account.

Apple Pay on the web requires that all participating websites register with Apple. After the domain is registered, domain name validation is performed only after Apple issues a TLS client certificate. Websites supporting Apple Pay are required to serve their content over HTTPS. For each payment transaction, websites need to obtain a secure and unique merchant session with an Apple server using the Apple-issued TLS client certificate. Merchant session data is signed by Apple. After a merchant session signature is verified, a website may query whether the user has an Apple Pay–capable device and whether they have a credit, debit, or prepaid card activated on the device. No other details are shared. If the user doesn't want to share this information, they can disable Apple Pay queries in Safari privacy settings on iPhone, iPad, and Mac devices.

After a merchant session is validated, all security and privacy measures are the same as when a user pays within an app.

If the user is transmitting payment-related information from a Mac to an iPhone or Apple Watch, Apple Pay Handoff uses the end-to-end encrypted Apple Identity Service (IDS) protocol to transmit payment-related information between the user's Mac and the authorizing device. IDS uses the user's device keys to perform encryption so no other device can decrypt this information, and the keys aren't available to Apple. Device discovery for Apple Pay Handoff contains the type and unique identifier of the user's credit cards along with some metadata. The device-specific account number of the user's card isn't shared, and it continues to remain stored securely on the user's iPhone or Apple Watch. Apple also securely transfers the user's recently used contact, shipping, and billing addresses over iCloud Keychain.

After the user authorizes payment using Touch ID, Face ID, a passcode, or double-clicking the side button on Apple Watch, a payment token uniquely encrypted to each website's merchant certificate is securely transmitted from the user's iPhone or Apple Watch to their Mac and then delivered to the merchant's website.

Only devices in proximity to each other may request and complete payment. Proximity is determined through Bluetooth Low Energy (BLE) advertisements.

Contactless passes in Apple Pay

To transmit data from supported passes to compatible NFC terminals, Apple uses the Apple Value Added Services (Apple VAS) protocol. The VAS protocol can be implemented on contactless terminals and uses NFC to communicate with supported Apple devices. The VAS protocol works over a short distance and can be used to present contactless passes independently or as part of an Apple Pay transaction.

When the device is held near the NFC terminal, the terminal initiates receiving the pass information by sending a request for a pass. If the user has a pass with the pass provider's identifier, the user is asked to authorize its use using Touch ID, Face ID, or a passcode. The pass information, a timestamp, and a single-use random ECDH P-256 key are used with the pass provider's public key to derive an encryption key for the pass data, which is sent to the terminal.

From iOS 12 to and including iOS 13, users may manually select a pass before presenting it to the merchant's NFC terminal. In iOS 13.1 or later, pass providers can configure manually selected passes to either require user authentication or be used without authentication.

Rendering cards unusable with Apple Pay

Credit, debit, and prepaid cards added to the Secure Element can be used only if the Secure Element is presented with authorization using the same pairing key and Authorization Random (AR) value from when the card was added. On receipt of a new AR value, the Secure Element marks any previously added cards as deleted. This allows the operating system to instruct the Secure Enclave to render cards unusable by marking its copy of the AR as invalid under the following scenarios:

Method	Device
When the passcode is disabled	iPhone, iPad, Apple Watch
When the password is disabled	Mac
The user signs out of iCloud	iPhone, iPad, Mac, Apple Watch
The user selects Erase All Content and Settings	iPhone, iPad, Apple Watch
The device is restored from Recovery mode	iPhone, iPad, Mac, Apple Watch
Unpairing	Apple Watch

Suspending, removing, and erasing cards

Users can suspend Apple Pay on iPhone, iPad, and Apple Watch by placing their devices in Lost Mode using Find My. Users also have the ability to remove and erase their cards from Apple Pay using Find My, iCloud.com, or directly on their devices using the Wallet app. On Apple Watch, cards can be removed using iCloud settings, the Apple Watch app on iPhone, or directly on the watch. The ability to make payments using cards on the device is suspended or removed from Apple Pay by the card issuer or respective payment network, even if the device is offline and not connected to a cellular or Wi-Fi network. Users can also call their card issuer to suspend or remove cards from Apple Pay.

When a user erases the entire device—using Erase All Content and Settings, using Find My, or restoring their device—iPhone, iPad, iPod touch, Mac, and Apple Watch instruct the Secure Element to mark all cards as deleted. This has the effect of immediately changing the cards to an unusable state until the Apple Pay servers can be contacted to fully erase the cards from the Secure Element. Independently, the Secure Enclave marks the AR as invalid so that further payment authorizations for previously enrolled cards aren't possible. When the device is online, it attempts to contact the Apple Pay servers to ensure that all cards in the Secure Element are erased.

Apple Cash security in iOS, iPadOS, and watchOS

Overview

In iOS 11.2 or later, iPadOS 13.1 or later, and watchOS 4.2 or later, Apple Pay can be used on an iPhone, iPad, or Apple Watch to send, receive, and request money from other users. When a user receives money, it's added to an Apple Cash account that can be accessed in the Wallet app or within Settings > Wallet & Apple Pay across any of the eligible devices the user has signed in with their Apple ID.

In iOS 14, iPadOS 14, and watchOS 7, the organizer of an iCloud family who has verified their identity can enable Apple Cash for their family members under the age of 18. Optionally, the organizer can restrict the money sending capabilities of these users to family members only or contacts only. If the family member under the age of 18 goes through an Apple ID account recovery, the organizer of the family must manually reenable the Apple Cash card for that user. If the family member under the age of 18 is no longer part of the iCloud family, their Apple Cash balance is automatically transferred to the organizer's account.

When the user sets up Apple Cash, the same information as when the user adds a credit or debit card may be shared with our partner bank Green Dot Bank and with Apple Payments Inc., a wholly owned subsidiary created to protect the user's privacy by storing and processing information separately from the rest of Apple, and in a way that the rest of Apple doesn't know. This information is used only for troubleshooting, fraud prevention, and regulatory purposes.

Using Apple Cash in iMessage

To use person-to-person payments and Apple Cash, a user must be signed in to their iCloud account on an Apple Cash–compatible device and have two-factor authentication set up on the iCloud account. Money requests and transfers between users are initiated from within the Messages app or by asking Siri. When a user attempts to send money, iMessage displays the Apple Pay sheet. The Apple Cash balance is always used first. If necessary, additional funds are drawn from a second credit or debit card the user has added to the Wallet app.

Using Apple Cash in stores, apps, and on the web

The Apple Cash card in the Wallet app can be used with Apple Pay to make payments in stores, in apps, and on the web. Money in the Apple Cash account can also be transferred to a bank account. In addition to money being received from another user, money can be added to the Apple Cash account from a debit or prepaid card in the Wallet app.

Apple Payments Inc. stores, and may use, the user's transaction data for troubleshooting, fraud prevention, and regulatory purposes once a transaction is completed. The rest of Apple doesn't know who the user sent money to, received money from, or where the user made a purchase with their Apple Cash card.

When the user sends money with Apple Pay, adds money to an Apple Cash account, or transfers money to a bank account, a call is made to the Apple Pay servers to obtain a cryptographic nonce, which is similar to the value returned for Apple Pay within apps. The nonce, along with other transaction data, is passed to the Secure Element to generate a payment signature. When the payment signature comes out of the Secure Element, it's passed to the Apple Pay servers. The authentication, integrity, and correctness of the transaction is verified through the payment signature and the nonce by Apple Pay servers. Money transfer is then initiated, and the user is notified of a completed transaction.

If the transaction involves:

- A debit card for adding money to Apple Cash
- Providing supplemental money if the Apple Cash balance is insufficient

An encrypted payment credential is also produced and sent to Apple Pay servers, similar to how Apple Pay works within apps and websites.

After the balance of the Apple Cash account exceeds a certain amount or if unusual activity is detected, the user is prompted to verify their identity. Information provided to verify the user's identity—such as social security number or answers to questions (for example, to confirm a street name the user lived on previously)—is securely transmitted to the Apple partner and encrypted using their key. Apple can't decrypt this data. The user is prompted to verify their identity again if they perform an Apple ID account recovery, before regaining access to their Apple Cash balance.

Apple Card security

Apple Card application in the Wallet app

In iOS 12.4 or later, macOS 10.14.6 or later, and watchOS 5.3 or later, Apple Card can be used with Apple Pay to make payments in stores, in apps, and on the web.

To apply for Apple Card, the user must be signed into their iCloud account on an Apple Pay–compatible iOS or iPadOS device and have two-factor authentication set up on the iCloud account. When the application is approved, Apple Card is available in the Wallet app or within Settings > Wallet & Apple Pay across any of the eligible devices the user has signed in with their Apple ID.

When a user applies for Apple Card, user identity information is securely verified by Apple's identity provider partners and then shared with Goldman Sachs Bank USA for the purposes of identity and credit evaluation.

Information such as the social security number or ID document image provided during the application is securely transmitted to Apple's identity provider partners and/or Goldman Sachs Bank USA encrypted with their respective keys. Apple can't decrypt this data.

The income information provided during the application, and the bank account information used for bill payments, are securely transmitted to Goldman Sachs Bank USA encrypted with their key. The bank account information is saved in the keychain. Apple can't decrypt this data.

When adding Apple Card to the Wallet app, the same information as when a user adds a credit or debit card may be shared with the Apple partner bank Goldman Sachs Bank USA and with Apple Payments Inc. This information is used only for troubleshooting, fraud prevention, and regulatory purposes.

A physical card can be ordered from Apple Card in the Wallet app. After the user receives the physical card, it's activated using the NFC tag present in the bifold envelope of the physical card. The tag is unique per card and can't be used to activate another user's card. Alternatively, the card can be manually activated in the Wallet settings. Additionally, the user can also choose to lock or unlock the physical card at any time from the Wallet app.

Apple Card payments and Apple Wallet pass details

Payments due on the Apple Card account can be made from the Wallet app in iOS with Apple Cash and a bank account. Bill payments can be scheduled as recurring or as a one-time payment at a specific date with Apple Cash and a bank account. When a user makes a payment, a call is made to the Apple Pay servers to obtain a cryptographic nonce similar to Apple Cash. The nonce, along with the payment setup details, is passed to the Secure Element to generate a signature. When the payment signature comes out of the Secure Element, it's passed to the Apple Pay servers. The authentication, integrity, and correctness of the payment are verified through the signature and the nonce by Apple Pay servers, and the order is passed on to Goldman Sachs Bank USA for processing.

Displaying the Apple Card number details in the pass using the Wallet app requires user authentication with Face ID, Touch ID, or a passcode. It can be replaced by the user in the card information section and disables the previous one.

Adding transit and student ID cards to Wallet

Transit cards

In many global markets, users can add supported transit cards to the Wallet app on supported models of iPhone and Apple Watch. Depending on the transit operator, this may be done by transferring the value and commuter pass from a physical card into its digital Apple Wallet representation or by provisioning a new transit card into the Wallet app from the Wallet app or the transit card issuer's app. After transit cards are added to the Wallet app, users can ride transit simply by holding their iPhone or Apple Watch near the transit reader. Some cards can also be used to make payments.

Added transit cards are associated with a user's iCloud account. If the user adds more than one card to the Wallet app, Apple or the transit card issuer may be able to link the user's personal information and the associated account information between cards. Transit cards and transactions are protected by a set of hierarchical cryptographic keys.

During the process of transferring the balance from a physical card to the Wallet app, users are required to enter card specific information. Users may also need to provide personal information for proof of card possession. When transferring passes from iPhone to Apple Watch, both devices must be online during transfer.

The balance can be recharged with funds from credit, debit and prepaid cards through Wallet or from the transit card issuer's app. To understand the security of reloading the balance when using Apple Pay, see Paying with cards within apps. To learn how the transit card is provisioned from within the transit card issuer's app, see Adding credit or debit cards from a card issuer's app.

If provisioning from a physical card is supported, the transit card issuer has the cryptographic keys needed to authenticate the physical card and verify the user's entered data. After the data is verified, the system can create a Device Account Number for the Secure Element and activate the newly added pass in the Wallet app with the transferred balance. In some cities, after provisioning from the physical card is complete, the physical card is disabled.

At the end of either type of provisioning, if the transit card balance is stored on the device, it's encrypted and stored to a designated applet in the Secure Element. The transit operator has the keys to perform cryptographic operations on the card data for balance transactions.

By default, users benefit from the seamless Express Transit experience that allows them to pay and ride without requiring Touch ID, Face ID, or a passcode. Information such as recently visited stations, transaction history, and additional tickets may be accessed by any nearby contactless card reader with Express Mode enabled. Users can enable the Touch ID, Face ID, or passcode authorization requirement in the Wallet & Apple Pay settings by disabling Express Transit.

As with other Apple Pay cards, users can suspend or remove transit cards by:

- Erasing the device remotely with Find My
- Enabling Lost Mode with Find My
- Entering a mobile device management (MDM) remote wipe command
- Removing all cards from their Apple ID account page
- Removing all cards from iCloud.com
- Removing all cards from the Wallet app
- Removing the card in the issuer's app

Apple Pay servers notify the transit operator to suspend or disable those cards. If a user removes a transit card from an online device, the balance can be recovered by adding it back to a device signed in with the same Apple ID. If a device is offline, powered off, or unusable, recovery may not be possible.

Credit and debit cards

In some cities, transit readers accept EMV (smart) cards to pay for transit rides. When users present an EMV credit or debit card to those readers, user authentication is required just as with "Pay with credit and debit cards in the stores."

In iOS 12.3 or later, some existing EMV credit/debit cards in the Wallet app can be enabled for Express Transit, which allows the user to pay for a trip at supported transit operators without requiring Touch ID, Face ID, or a passcode. When a user provisions an EMV credit or debit card, the first card provisioned to the Wallet app is enabled for Express Transit. The user can tap the More button on the front of the card in the Wallet app and disable Express Transit for that card by setting Express Transit Settings to None. The user can also select a different credit or debit card as their Express Transit card via the Wallet app. Touch ID, Face ID, or a passcode is required to reenable or select a different card for Express Transit.

Apple Card and Apple Cash are eligible for Express Transit.

Student ID cards

In iOS 12 or later, students, faculty, and staff at participating campuses can add their student ID card to the Wallet app on supported models of iPhone and Apple Watch to access locations and pay wherever their card is accepted.

A user adds their student ID card to the Wallet app through an app provided by the card issuer or participating school. The technical process by which this occurs is the same as the one described in Adding credit or debit cards from a card issuer's app. In addition, issuing apps must support two-factor authentication on the accounts that guard access to their student IDs. A card may be set up simultaneously on up to any two supported Apple devices signed in with the same Apple ID.

When a student ID card is added to the Wallet app, Express Mode is turned on by default. Student ID cards in Express Mode interact with accepting terminals without Touch ID, Face ID, passcode authentication, or double-clicking the side button on Apple Watch. The user can tap the More button on the front of the card in the Wallet app and turn off Express Mode to disable this feature. Touch ID, Face ID, or a passcode is required to reenable Express Mode.

Student ID cards can be disabled or removed by:

- Erasing the device remotely with Find My
- Enabling Lost Mode with Find My
- Receiving a mobile device management (MDM) remote wipe command
- Removing all cards from their Apple ID account page
- Removing all cards from iCloud.com
- Removing all cards from the Wallet app
- Removing the card in the issuer's app

iMessage

iMessage security overview

Apple iMessage is a messaging service for iOS and iPadOS devices, Apple Watch, and Mac computers. iMessage supports text and attachments such as photos, contacts, locations, links, and attachments directly on to a message, such as a thumbs up icon. Messages appear on all of a user's registered devices so that a conversation can be continued from any of the user's devices. iMessage makes extensive use of the Apple Push Notification service (APNs). Apple doesn't log the contents of messages or attachments, which are protected by end-to-end encryption so no one but the sender and receiver can access them. Apple can't decrypt the data.

When a user turns on iMessage on a device, the device generates encryption and signing pairs of keys for use with the service. For encryption, there is an encryption RSA 1280-bit key as well as an encryption EC 256-bit key on the NIST P-256 curve. For signatures, Elliptic Curve Digital Signature Algorithm (ECDSA) 256-bit signing keys are used. The private keys are saved in the device's keychain and only available after first unlock. The public keys are sent to Apple Identity Service (IDS), where they are associated with the user's phone number or email address, along with the device's APNs address.

As users enable additional devices for use with iMessage, their encryption and signing public keys, APNs addresses, and associated phone numbers are added to the directory service. Users can also add more email addresses, which are verified by sending a confirmation link. Phone numbers are verified by the carrier network and SIM. With some networks, this requires using SMS (the user is presented with a confirmation dialog if the SMS isn't zero rated). Phone number verification may be required for several system services in addition to iMessage, such as FaceTime and iCloud. All of the user's registered devices display an alert message when a new device, phone number, or email address is added.

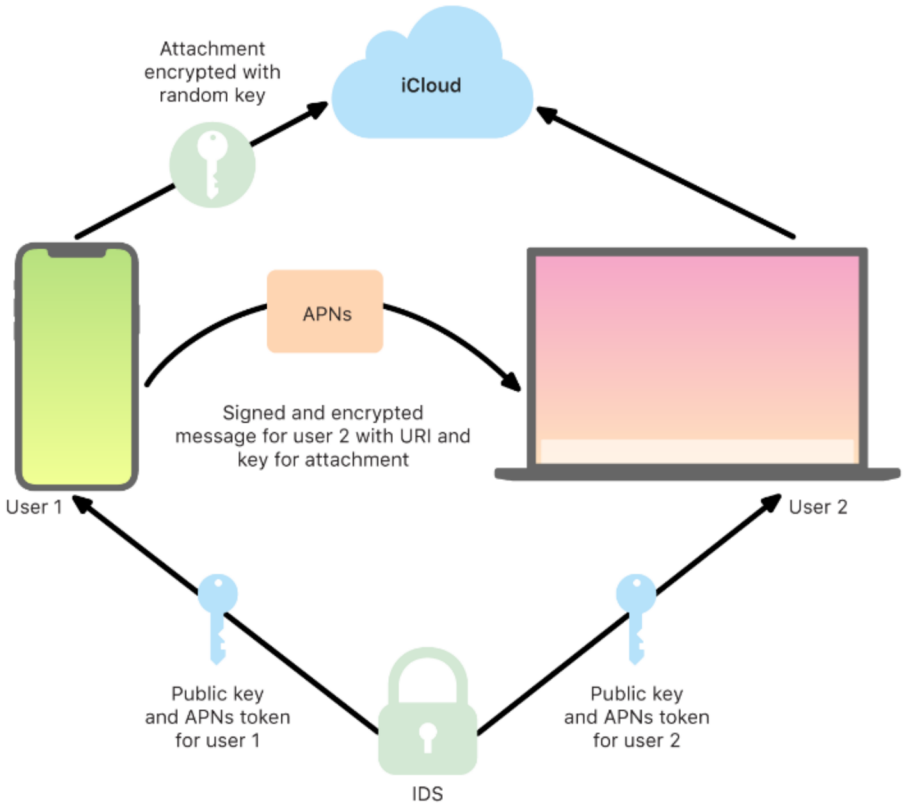
How iMessage sends and receives messages securely

Users start a new iMessage conversation by entering an address or name. If they enter a phone number or email address, the device contacts the Apple Identity Service (IDS) to retrieve the public keys and APNs addresses for all of the devices associated with the addressee. If the user enters a name, the device first uses the user's Contacts app to gather the phone numbers and email addresses associated with that name and then gets the public keys and APNs addresses from IDS.

The user's outgoing message is individually encrypted for each of the receiver's devices. The public encryption keys and signing keys of the receiving devices are retrieved from IDS. For each receiving device, the sending device generates a random 88-bit value and uses it as an HMAC-SHA256 key to construct a 40-bit value derived from the sender and receiver public key and the plaintext. The concatenation of the 88-bit and 40-bit values makes a 128-bit key, which encrypts the message with it using AES in Counter (CTR) Mode. The 40-bit value is used by the receiver side to verify the integrity of the decrypted plaintext. This per-message AES key is encrypted using RSA-OAEP to the public key of the receiving device. The combination of the encrypted message text and the encrypted message key is then hashed with SHA-1, and the hash is signed with the Elliptic Curve Digital Signature Algorithm (ECDSA) using the sending device's private signing key. In iOS 13 or later and iPadOS 13.1 or later, devices may use an Elliptic Curve Integrated Encryption Scheme (ECIES) encryption instead of RSA encryption.

The resulting messages, one for each receiving device, consist of the encrypted message text, the encrypted message key, and the sender's digital signature. They are then dispatched to the APNs for delivery. Metadata, such as the timestamp and APNs routing information, isn't encrypted. Communication with APNs is encrypted using a forward-secret TLS channel.

APNs can only relay messages up to 4 or 16 KB in size, depending on the iOS or iPadOS version. If the message text is too long or if an attachment such as a photo is included, the attachment is encrypted using AES in CTR mode with a randomly generated 256-bit key and uploaded to iCloud. The AES key for the attachment, its Uniform Resource Identifier (URI), and an SHA-1 hash of its encrypted form are then sent to the recipient as the contents of an iMessage, with their confidentiality and integrity protected through normal iMessage encryption, as shown in the following diagram.



How iMessage sends and receives messages.

For group conversations, this process is repeated for each recipient and their devices.

On the receiving side, each device receives its copy of the message from APNs and, if necessary, retrieves the attachment from iCloud. The incoming phone number or email address of the sender is matched to the receiver's contacts so that a name can be displayed when possible.

As with all push notifications, the message is deleted from APNs when it's delivered. Unlike other APNs notifications, however, iMessage messages are queued for delivery to offline devices. Messages are stored for up to 30 days.

Secure iMessage name and photo sharing

iMessage Name and Photo Sharing allows a user to share a Name and Photo using iMessage. The user may select their Me Card information, or customize the name and include any image they choose. iMessage Name and Photo sharing uses a two-stage system to distribute the name and photo.

The data is subdivided in fields, each encrypted and authenticated separately as well as authenticated together with the process below. There are three fields:

- Name
- Photo
- Photo filename

A first step of the data creation is to randomly generate a record 128-bit key on the device. This record key is then derived with HKDF-HMAC-SHA256 to create three subkeys: Key 1:Key 2:Key 3 = HKDF(record key, "nicknames"). For each field, a random 96-bit Initialization Vector (IV) is generated and the data is encrypted using AES-CTR and Key 1. A message authentication code (MAC) is then computed with HMAC-SHA256 using Key 2 and covering the field name, the field IV, and the field ciphertext. Finally, the set of individual field MAC values are concatenated and their MAC is computed with HMAC-SHA256 using Key 3. The 256-bit MAC is stored along side the encrypted data. The first 128 bits of this MAC is used as RecordID.

This encrypted record is then stored in the CloudKit public database under the RecordID. This record is never mutated, and when the user chooses to change their name and photo, a new encrypted record is generated each time. When user 1 chooses to share their name and photo with user 2, they send the record key along with the recordID inside their iMessage payload, which is encrypted.

When user 2's device receives this iMessage payload, it notices that the payload contains a Nickname and Photo recordID and key. User 2's device then goes out to the public CloudKit database to retrieve the encrypted name and photo at the record ID and sends it across using iMessage.

After the message is retrieved, user 2's device decrypts the payload and verifies the signature using the recordID itself. If this passes, user 2 is presented with the name and photo and they can choose to add this to their contacts, or use it for Messages.

Secure Business Chat using the Messages app

Business Chat is a messaging service that enables users to communicate with businesses using the Messages app. With Business Chat, the user is always in control of the conversation. They can also delete the conversation and block the business from messaging them in the future. For privacy, the business doesn't receive the user's phone number, email address, or iCloud account information. Instead, a custom unique identifier called the *Opaque ID* is generated by the Apple Identity Service (IDS) and shared with the business. The Opaque ID is unique to the relationship between the user's Apple ID and the business's Business ID. A user has a different Opaque ID for every business they contact using Business Chat. The user decides if and when to share personal identifying information with the business.

Business Chat supports Managed Apple IDs from Apple Business Manager and determines whether they are enabled for iMessage and FaceTime in Apple School Manager.

Messages sent to the business are encrypted between the user's device and Apple's messaging servers, using the same security and Apple messaging servers as iMessages. Apple messaging servers decrypt these messages in RAM, and relay them to the business over an encrypted link using TLS 1.2. Messages are never stored in unencrypted form while transiting through Apple's Business Chat service. Businesses' replies are also sent using TLS 1.2 to the Apple messaging servers, where they are encrypted using the unique public keys of each recipient device.

If user devices are online, the message is delivered immediately and isn't cached on the Apple messaging servers. If a user's device isn't online, the encrypted message is cached for up to 30 days to enable the user to receive it when the device is back online. As soon as the device is back online, the message is delivered and deleted from cache. After 30 days, an undelivered cached message expires and is permanently deleted.

The Business Chat service never stores conversation history.

FaceTime security

FaceTime is Apple's video and audio calling service. Like iMessage, FaceTime calls use the Apple Push Notification service (APNs) to establish an initial connection to the user's registered devices. The audio/video contents of FaceTime calls are protected by end-to-end encryption, so no one but the sender and receiver can access them. Apple can't decrypt the data.

The initial FaceTime connection is made through an Apple server infrastructure that relays data packets between the users' registered devices. Using APNs notifications and Session Traversal Utilities for NAT (STUN) messages over the relayed connection, the devices verify their identity certificates and establish a shared secret for each session. The shared secret is used to derive session keys for media channels streamed using the Secure Real-time Transport Protocol (SRTP). SRTP packets are encrypted using AES256 in Counter Mode and HMAC-SHA1. Subsequent to the initial connection and security setup, FaceTime uses STUN and Internet Connectivity Establishment (ICE) to establish a peer-to-peer connection between devices, if possible.

Group FaceTime extends FaceTime to support up to 33 concurrent participants. As with classic one-to-one FaceTime, calls are end-to-end encrypted among the invited participants' devices. Even though Group FaceTime reuses much of the infrastructure and design of one-to-one FaceTime, Group FaceTime calls feature a new key-establishment mechanism built on top of the authenticity provided by Apple Identity Service (IDS). This protocol provides forward secrecy, meaning the compromise of a user's device won't leak the contents of past calls. Session keys are wrapped using AES-SIV and distributed among participants using an ECIES construction with ephemeral P-256 ECDH keys.

When a new phone number or email address is added to an ongoing Group FaceTime call, active devices establish new media keys and never share previously used keys with the newly invited devices.

Find My

Find My security

Overview

The Find My app combines Find My iPhone and Find My Friends into a single app in iOS, iPadOS, and macOS. Find My can help users locate a missing device, even an offline Mac. An online device can simply report its location to the user via iCloud. Find My works offline by sending out short range Bluetooth signals from the missing device that can be detected by other Apple devices in use nearby. Those nearby devices then relay the detected location of the missing device to iCloud so users can locate it in the Find My app—all while protecting the privacy and security of all the users involved. Find My even works with a Mac that is offline and asleep.

Using Bluetooth and the hundreds of millions of iOS, iPadOS, and macOS devices in active use around the world, the user can locate a missing device even if it can't connect to a Wi-Fi or cellular network. Any iOS, iPadOS, or macOS device with "offline finding" enabled in Find My settings can act as a "finder device." This means the device can detect the presence of another missing offline device using Bluetooth and then use its network connection to report an approximate location back to the owner. When a device has offline finding enabled, it also means that it can be located by other participants in the same way. This entire interaction is end-to-end encrypted, anonymous, and designed to be battery and data efficient, so there is minimal impact on battery life cellular data plan usage and user privacy is protected.

Note: Find My may not be available in all countries or regions.

End-to-end encryption

Find My is built on a foundation of advanced public key cryptography. When offline finding is enabled in Find My settings, an elliptic curve (EC) P-224 private encryption key pair noted {d,P} is generated directly on the device where *d* is the private key and *P* is the public key. Additionally, a 256-bit secret SK_0 and a counter $_i$ is initialized to zero. This private key pair and the secret are never sent to Apple and are synced only among the user's other devices in an end-to-end encrypted manner using iCloud Keychain. The secret and the counter are used to derive the current symmetric key SK_i with the following recursive construction: $SK_i = KDF(SK_{i-1}, \text{"update"})$

Based on the key SK_i, two large integers u_i and v_i are computed with $(u_i,v_i) = KDF(SK_i, \text{"diversify"})$. Both the P-224 private key denoted *d* and corresponding public key referred to as *P* are then derived using an affine relation involving the two integers to compute a short-lived key pair: The derived private key is d_i, where $d_i = u_i * d + v_i$ (modulo the order of the P-224 curve) and the corresponding public part is P_i and verifies that $P_i = u_i*P + v_i*G$.

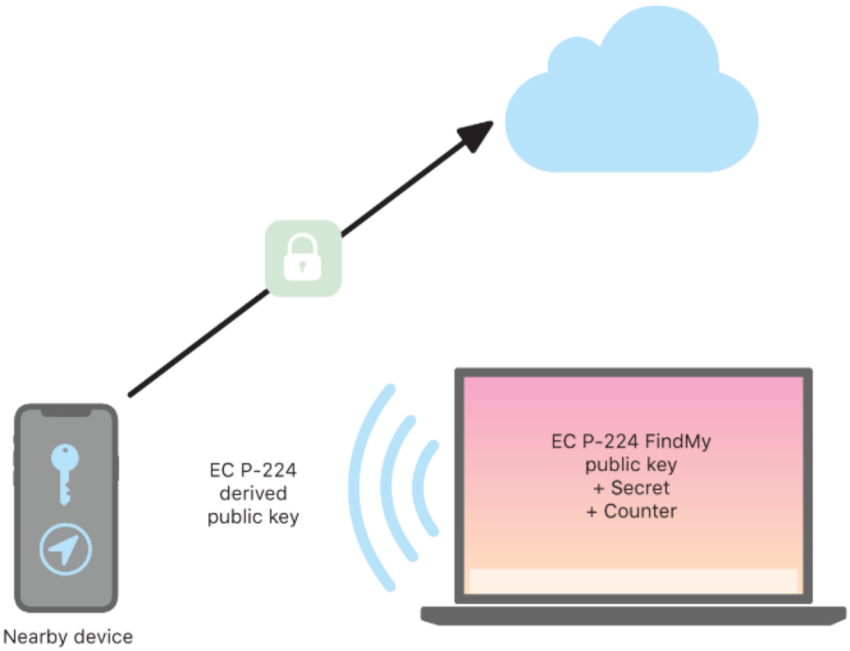
When a device goes missing and can't connect to Wi-Fi or cellular—for example, a MacBook Pro left on a park bench—it begins periodically broadcasting the derived public key P_i for a limited period of time in a Bluetooth payload. By using P-224, the public key representation can fit into a single Bluetooth payload. The surrounding devices can then help in the finding of the offline device by encrypting their location to the public key. Approximately every 15 minutes, the public key is replaced by a new one using an incremented value of the counter and the process above so that the user can't be tracked by a persistent identifier. The derivation mechanism prevents the various public keys P_i from being linked to the same device.

Keeping users and devices anonymous

In addition to making sure that location information and other data are fully encrypted, participants' identities remain private from each other and from Apple. The traffic sent to Apple by finder devices contains no authentication information in the contents or headers. As a result, Apple doesn't know who the finder is or whose device has been found. Further, Apple doesn't log information that would reveal the identity of the finder and retains no information that would allow anyone to correlate the finder and owner. The device owner receives only the encrypted location information that's decrypted and displayed in the Find My app with no indication as to who found the device.

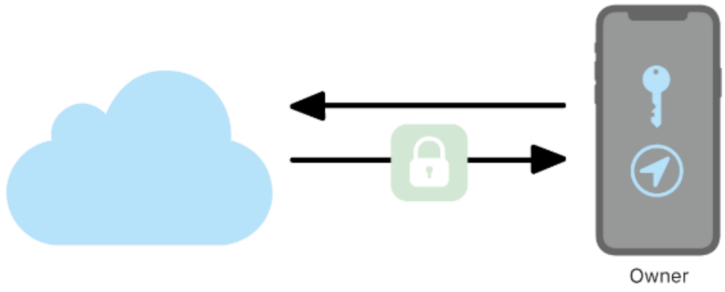
Using Find My to locate missing Apple devices

Any Apple devices within Bluetooth range that have offline finding enabled can detect a signal from another Apple device configured to allow Find My and read the current broadcast key P_i. Using an ECIES construction and the public key P_i from the broadcast, the finder devices encrypt their current location information and relay it to Apple. The encrypted location is associated with a server index which is computed as the SHA256 hash of the P-224 public key P_i obtained from the Bluetooth payload. Apple never has the decryption key, so Apple can't read the location encrypted by the finder. The owner of the missing device can reconstruct the index and decrypt the encrypted location.



How Find My locates devices.

When trying to locate the missing device, an expected range of counter values is estimated for the location search period. With the knowledge of the original private P-224 key *d* and secret values SK_i in the range of counter values of the search period the owner can then reconstruct the set of values $\{d_i, SHA256(P_i)\}$ for the entire search period. The owner device used to locate the missing device can then perform queries to the server using the set of index values $SHA256(P_i)$ and download the encrypted locations from the server. The Find My app then locally decrypts the encrypted locations with the matching private keys d_i and shows an approximate location of the missing device in the app. Location reports from multiple finder devices are combined by the owner's app to generate a more precise location.



How the owner gets the device location from the Find My app.

Locating devices that are offline

If a user has Find My iPhone enabled on their device, offline finding is enabled by default when they upgrade a device to iOS 13 or later, iPadOS 13.1 or later, and macOS 10.15 or later. This ensures that every user has the best possible chance to locate their device if it goes missing. However, if at any time the user prefers not to participate, they can disable offline finding in Find My settings on their device. When offline finding is disabled, the device no longer acts as a finder nor is it detectable by other finder devices. However, the user can still locate the device as long as it can connect to a Wi-Fi or cellular network.

When a missing offline device is located, the user receives a notification and email message to let them know the device has been found. To view the location of the missing device, the user opens the Find My app and selects the Devices tab. Rather than showing the device on a blank map as it would have prior to the device being located, Find My shows a map location with an approximate address and information on how long ago the device was detected. If more location reports come in, the current location and time stamp both update automatically. Although users can't play a sound on an offline device or erase it remotely, they can use the location information to retrace their steps or take other actions to help them recover it.

Continuity

Continuity security overview

Continuity takes advantage of technologies like iCloud, Bluetooth, and Wi-Fi to enable users to continue an activity from one device to another, make and receive phone calls, send and receive text messages, and share a cellular internet connection.

Handoff security

Overview

With Handoff, when a user's iOS, iPadOS, and macOS devices are near each other, the user can automatically pass whatever they're working on from one device to the other. Handoff lets the user switch devices and instantly continue working.

When a user signs in to iCloud on a second Handoff-capable device, the two devices establish a Bluetooth Low Energy (BLE) 4.2 pairing out-of-band using APNs. The individual messages are encrypted much like messages in iMessage are. After the devices are paired, each device generates a symmetric 256-bit AES key that gets stored in the device's keychain. This key can encrypt and authenticate the BLE advertisements that communicate the device's current activity to other iCloud paired devices using AES256 in GCM mode, with replay protection measures.

The first time a device receives an advertisement from a new key, it establishes a BLE connection to the originating device and performs an advertisement encryption key exchange. This connection is secured using standard BLE 4.2 encryption as well as encryption of the individual messages, which is similar to how iMessage is encrypted. In some situations, these messages are sent using APNs instead of BLE. The activity payload is protected and transferred in the same way as an iMessage.

Handoff between native apps and websites

Handoff allows an iOS, iPadOS, or macOS native app to resume user activity on a webpage in domains legitimately controlled by the app developer. It also allows the native app user activity to be resumed in a web browser.

To prevent native apps from claiming to resume websites not controlled by the developer, the app must demonstrate legitimate control over the web domains it wants to resume. Control over a website domain is established using the mechanism for shared web credentials. For details, see App access to saved passwords. The system must validate an app's domain name control before the app is permitted to accept user activity Handoff.

The source of a webpage Handoff can be any browser that has adopted the Handoff APIs. When the user views a webpage, the system advertises the domain name of the webpage in the encrypted Handoff advertisement bytes. Only the user's other devices can decrypt the advertisement bytes.

On a receiving device, the system detects that an installed native app accepts Handoff from the advertised domain name and displays that native app icon as the Handoff option. When launched, the native app receives the full URL and the title of the webpage. No other information is passed from the browser to the native app.

In the opposite direction, a native app may specify a fallback URL when a Handoff receiving device doesn't have the same native app installed. In this case, the system displays the user's default browser as the Handoff app option (if that browser has adopted Handoff APIs). When Handoff is requested, the browser is launched and given the fallback URL provided by the source app. There is no requirement that the fallback URL be limited to domain names controlled by the native app developer.

Handoff of larger data

In addition to using the basic feature of Handoff, some apps may elect to use APIs that support sending larger amounts of data over Apple-created peer-to-peer Wi-Fi technology (much like AirDrop). For example, the Mail app uses these APIs to support Handoff of a mail draft, which may include large attachments.

When an app uses this facility, the exchange between the two devices starts off just as in Handoff. However, after receiving the initial payload using Bluetooth Low Energy (BLE), the receiving device initiates a new connection over Wi-Fi. This connection is encrypted (with TLS), which exchanges their iCloud identity certificates. The identity in the certificates is verified against the user's identity. Further payload data is sent over this encrypted connection until the transfer is complete.

Universal Clipboard

Universal Clipboard leverages Handoff to securely transfer the content of a user's clipboard across devices so they can copy on one device and paste on another. Content is protected in the same way as other Handoff data and is shared by default with Universal Clipboard unless the app developer chooses to disallow sharing.

Apps have access to clipboard data regardless of whether the user has pasted the clipboard into the app. With Universal Clipboard, this data access extends to apps on the user's other devices (as established by their iCloud sign-in).

iPhone cellular call relay security

When a user's Mac, iPad, iPod touch, or HomePod is on the same Wi-Fi network as their iPhone, it can make and receive phone calls using the cellular connection on iPhone. Configuration requires the devices to be signed in to both iCloud and FaceTime using the same Apple ID account.

When an incoming call arrives, all configured devices are notified using the Apple Push Notification service (APNs), with each notification using the same end-to-end encryption as iMessage. Devices that are on the same network present the incoming call notification user interface. When the user answers the call, the audio is seamlessly transmitted from the user's iPhone using a secure peer-to-peer connection between the two devices.

When a call is answered on one device, ringing of nearby iCloud-paired devices is terminated by briefly advertising using Bluetooth Low Energy (BLE). The advertising bytes are encrypted using the same method as Handoff advertisements.

Outgoing calls are also relayed to iPhone using APNs, and audio is similarly transmitted over the secure peer-to-peer link between devices. Users can disable phone call relay on a device by turning off iPhone Cellular Calls in FaceTime settings.

iPhone Text Message Forwarding security

Text Message Forwarding automatically sends SMS text messages received on an iPhone to a user's enrolled iPad, iPod touch, or Mac. Each device must be signed in to the iMessage service using the same Apple ID account. When Text Message Forwarding is turned on, enrollment is automatic on devices within a user's circle of trust if two-factor authentication is enabled. Otherwise, enrollment is verified on each device by entering a random six-digit numeric code generated by iPhone.

After devices are linked, iPhone encrypts and forwards incoming SMS text messages to each device, utilizing the methods described in iMessage security overview. Replies are sent back to iPhone using the same method, and then iPhone sends the reply as a text message using the carrier's SMS transmission mechanism. Text Message Forwarding can be turned on or off in Messages settings.

Instant Hotspot security

Instant Hotspot connects other Apple devices to a personal iOS or iPadOS hotspot. iOS and iPadOS devices that support Instant Hotspot use Bluetooth Low Energy (BLE) to discover and communicate to all devices that have signed in to the same individual iCloud account or accounts used with Family Sharing (in iOS 13 and iPadOS). Compatible Mac computers with OS X 10.10 or later use the same technology to discover and communicate with Instant Hotspot iOS and iPadOS devices.

Initially, when a user enters Wi-Fi settings on a device, it emits a BLE advertisement containing an identifier that all devices signed in to the same iCloud account agree upon. The identifier is generated from a DSID (Destination Signaling Identifier) that's tied to the iCloud account and rotated periodically. When other devices signed in to the same iCloud account are in close proximity and support Personal Hotspot, they detect the signal and respond, indicating the availability to use Instant Hotspot.

When a user who isn't part of Family Sharing chooses an iPhone or iPad for Personal Hotspot, a request to turn on Personal Hotspot is sent to that device. The request is sent across a link that is encrypted using BLE encryption, and the request is encrypted in a fashion similar to iMessage encryption. The device then responds across the same BLE link using the same per-message encryption with Personal Hotspot connection information.

For users that are part of Family Sharing, Personal Hotspot connection information is securely shared using a mechanism similar to that used by HomeKit devices to sync information. Specifically, the connection that shares hotspot information between users is secured with an ECDH (Curve25519) ephemeral key that is authenticated with the users' respective device-specific Ed25519 public keys. The public keys used are those that had previously synced between the members of Family Sharing using IDS when the Family Share was established.

Car keys security in iOS

Overview

The car keys feature is supported natively in supported iPhone devices and paired Apple Watch devices. Car keys are represented as passes (created by Apple on behalf of the automaker) in the Wallet app and support the full Apple Pay card life cycle (iCloud Lost Mode, Remote Wipe, local pass deletion, and Erase All Content and Settings). In addition to the standard Apple Pay card management, shared car keys can be deleted from the owner's iPhone, Apple Watch, and in the vehicle's Human Machine Interface (HMI).

Car keys can be used to unlock and lock the vehicle and to start the engine or set the vehicle into drive mode. The "standard transaction" offers mutual authentication and is mandatory for engine start. Unlock and lock transactions might use the "fast transaction" when required for performance reasons.

Keys are created through pairing an iPhone with an owned and supported vehicle. All keys are created on the embedded Secure Element based on elliptic curve (NIST P-256) on-board key generation (ECC-OBKG), and the private keys never leave the Secure Element. Communication between devices and the vehicle use the NFC standard, and key management uses an Apple to automaker server API with mutually authenticated TLS. After a key is paired to an iPhone, any Apple Watch paired to that iPhone can also receive a key. When a key is deleted either in the vehicle or on the device, it can't be restored. Keys on lost or stolen devices can be suspended and resumed, but reprovisioning them on a new device requires a new pairing or sharing.

Owner pairing

The owner must prove possession of the vehicle (the method is dependent on the automaker) and can start the pairing process in the automaker's app using an email link received from the automaker or from the vehicle menu. In all cases, the owner must present a confidential one-time pairing password to the iPhone, which is used to generate a secure pairing channel using the SPAKE2+ protocol with the NIST P-256 curve. When using the app or the email link, the password is automatically transferred to the iPhone where it must be entered manually when pairing is started from the vehicle.

Key sharing

The owner's paired iPhone can share keys to eligible family members' and friends' iPhone devices (and their paired Apple Watch devices) by sending a device-specific invitation using iMessage and the Apple Identity Service (IDS). All sharing commands are exchanged using the end-to-end encrypted IDS feature. The owner's paired iPhone keeps the IDS channel from changing during the sharing process.

Upon acceptance of the invitation, the family member's or friend's iPhone creates a digital key and sends the key creation certificate chain back to the owner's paired iPhone to verify that the key was created on an authentic Apple device. The owner's paired iPhone signs the ECC-public key of the other family member's or friend's iPhone and sends the signature back to the family member's or friend's iPhone. The signing operation in the owner device requires user authentication (Touch ID, Face ID or passcode entry) and a secure user intent described in Uses for Touch ID and Face ID. The authorization is requested when sending the invitation and is stored in the secure element for consumption when the friend device sends back the signing request.

Key deletion

Keys can be deleted on the keyholder device from the owner device and in the vehicle. Deletions on the keyholder iPhone are effective immediately, even if the keyholder uses the key. Therefore a strong warning is shown before the deletion.

Deletions of keys in the vehicle depends on whether the automaker requires the vehicle to be online for the deletion or not.

In both cases, the deletion on keyholder device or vehicle is reported to a key inventory server (KIS) on the automaker side, which registers issued keys for a vehicle for insurance purposes.

The owner can request a deletion from the back of the owner pass. The request is first sent to the automaker for key removal in the vehicle. The conditions for removing the key from the vehicle are defined by the automaker. Only when the key is removed in the vehicle will the automaker server send a remote termination request to the keyholder device.

When a key is terminated in a device, the applet that manages the digital car keys creates a cryptographically signed termination attestation, which is used as proof of deletion by the automaker and used to remove the key from the KTS.

Standard transactions

A secure channel between the reader and an iPhone is initiated by generating ephemeral key pairs on the reader and the iPhone side. Using a key agreement method, a shared secret can be derived on both sides and used for generation of a shared symmetric key using Diffie-Hellman, a key derivation function, and signatures from the long-term key established during pairing.

The ephemeral public key generated on the vehicle side is signed with the reader's long-term private key, which results in an authentication of the reader by the iPhone. From the iPhone perspective, this protocol is designed to prevent privacy-sensitive data from being revealed to an adversary intercepting the communication.

Finally, the iPhone uses the established secure channel to encrypt its public key identifier along with the signature computed on a reader's data-derived challenge and some additional app-specific data. This verification of the iPhone signature by the reader allows the reader to authenticate the device.

Fast transactions

The iPhone generates a cryptogram based on a secret previously shared during a standard transaction. This cryptogram allows the vehicle to quickly authenticate the device in performance sensitive scenarios. Optionally, a secure channel between the vehicle and the device is established by deriving session keys from a secret previously shared during a standard transaction and a new ephemeral key pair. The ability of the vehicle to establish the secure channel authenticates the vehicle to the iPhone.

Privacy

The key tracking server of the automaker doesn't store the device ID, SEID, or Apple ID. It stores only a mutable identifier—the instance CA identifier. This identifier isn't bound to any private data in the device or by the server, and it's deleted when the user wipes their device completely (using Erase All Contents and Settings).

Network security

Network security overview

In addition to the built-in safeguards Apple uses to protect data stored on Apple devices, there are many measures organizations can take to keep information secure as it travels to and from a device. All of these safeguards and measures fall under network security.

Users must be able to access corporate networks from anywhere in the world, so it's important to ensure that they are authorized and that their data is protected during transmission. To accomplish these security objectives, iOS, iPadOS, and macOS integrate proven technologies and the latest standards for both Wi-Fi and cellular data network connections. That's why our operating systems use—and provide developer access to—standard networking protocols for authenticated, authorized, and encrypted communications.

TLS security

iOS, iPadOS, and macOS support Transport Layer Security (TLS 1.0, TLS 1.1, TLS 1.2, TLS 1.3) and Datagram Transport Layer Security (DTLS). The TLS protocol supports both AES128 and AES256, and prefers cipher suites with forward secrecy. Internet apps such as Safari, Calendar, and Mail automatically use this protocol to enable an encrypted communication channel between the device and network services. High-level APIs (such as CFNetwork) make it easy for developers to adopt TLS in their apps, while low-level APIs (such as Network.framework) provide fine-grained control. CFNetwork disallows SSL 3, and apps that use WebKit (such as Safari) are prohibited from making an SSL 3 connection.

In iOS 11 or later and macOS 10.13 or later, SHA-1 certificates are no longer allowed for TLS connections unless trusted by the user. Certificates with RSA keys shorter than 2048 bits are also disallowed. The RC4 symmetric cipher suite is deprecated in iOS 10 and macOS 10.12. By default, TLS clients or servers implemented with SecureTransport APIs don't have RC4 cipher suites enabled and are unable to connect when RC4 is the only cipher suite available. To be more secure, services or apps that require RC4 should be upgraded to use secure cipher suites. In iOS 12.1, certificates issued after October 15, 2018, from a system-trusted root certificate must be logged in a trusted Certificate Transparency log to be allowed for TLS connections. In iOS 12.2, TLS 1.3 is enabled by default for Network.framework and NSURLSession APIs. TLS clients using the SecureTransport APIs can't use TLS 1.3.

App Transport Security

App Transport Security provides default connection requirements so that apps adhere to best practices for secure connections when using NSURLConnection, CFURL, or NSURLSession APIs. By default, App Transport Security limits cipher selection to include only suites that provide forward secrecy, specifically:

- ECDHE_ECDSA_AES and ECDHE_RSA_AES in Galois/Counter Mode (GCM)
- Cipher Block Chaining (CBC) mode

Apps are able to disable the forward secrecy requirement per domain, in which case RSA_AES is added to the set of available ciphers.

Servers must support TLS 1.2 and forward secrecy, and certificates must be valid and signed using SHA256 or stronger with a minimum 2048-bit RSA key or 256-bit elliptic curve key.

Network connections that don't meet these requirements will fail unless the app overrides App Transport Security. Invalid certificates always result in a hard failure and no connection. App Transport Security is automatically applied to apps that are compiled for iOS 9 or later and macOS 10.11 or later.

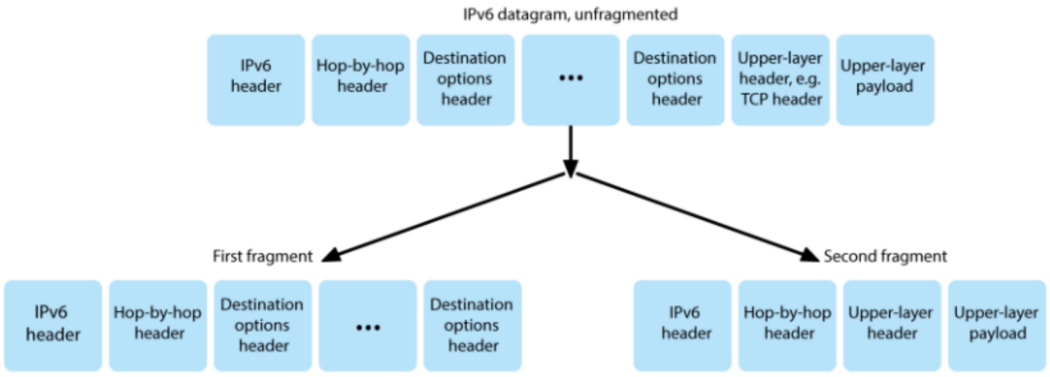
Certificate validity checking

Evaluating the trusted status of a TLS certificate is performed in accordance with established industry standards, as set out in RFC 5280, and incorporates emerging standards such as RFC 6962 (Certificate Transparency). In iOS 11 or later and macOS 10.13 or later, Apple devices are periodically updated with a current list of revoked and constrained certificates. The list is aggregated from certificate revocation lists (CRLs), which are published by each of the built-in root certificate authorities trusted by Apple, as well as by their subordinate CA issuers. The list may also include other constraints at Apple's discretion. This information is consulted whenever a network API function is used to make a secure connection. If there are too many revoked certificates from a CA to list individually, a trust evaluation may instead require that an online certificate status response (OCSP) is needed, and if the response isn't available, the trust evaluation will fail.

IPv6 security

All Apple operating systems support IPv6, implementing several mechanisms to protect the privacy of users and the stability of the networking stack. When Stateless Address Autoconfiguration (SLAAC) is used, the IPv6 addresses of all interfaces are generated in a way that prevents tracking devices across networks and at the same time allows for a good user experience by ensuring address stability when no network changes take place. The address generation algorithm is based on cryptographically generated addresses as of RFC 3972, enhanced by an interface-specific modifier to warrant that even different interfaces on the same network eventually have different addresses. Furthermore, temporary addresses are created with a preferred lifetime of 24 hours, and these are used by default for any new connections. Aligned with the Private Wi-Fi address feature introduced in iOS 14, iPadOS 14, and watchOS 7,a unique link-local address is generated for every Wi-Fi network that a device joins. The network's SSID is incorporated as an additional element for the address generation, similar to the Network_ID parameter as of RFC 7217. This approach is used in iOS 14, iPadOS 14, and watchOS 7.

To protect against attacks based on IPv6 extension headers and fragmentation, Apple devices implement protection measures specified in RFC 6980, RFC 7112, and RFC 8021. Among other measures, these inhibit attacks where the upper-layer header can be found only in the second fragment (as shown below), which in turn could cause ambiguities for security controls like stateless packet filters.



An IPv6 datagram.

In addition, to ensure the reliability of the iPv6 stack of Apple operating systems, Apple devices enforce various limits on IPv6-related data structures, such as the number of prefixes per interface.

Virtual private network (VPN) security

Secure network services like virtual private networking typically require minimal setup and configuration to work with iOS, iPadOS, and macOS devices.

Protocols supported

These devices work with VPN servers that support the following protocols and authentication methods:

- IKEv2/IPsec with authentication by shared secret, RSA Certificates, Elliptic Curve Digital Signature Algorithm (ECDSA) Certificates, EAP-MSCHAPv2, or EAP-TLS
- SSL-VPN using the appropriate client app from the App Store
- L2TP/IPsec with user authentication by MS-CHAPV2 password and machine authentication by shared secret (iOS, iPadOS, and macOS) and RSA SecurID or CRYPTOCard (macOS only)
- Cisco IPsec with user authentication by password, RSA SecurID or CRYPTOCard, and machine authentication by shared secret and certificates (macOS only)

VPN deployments supported

iOS, iPadOS, and macOS support the following:

- *VPN On Demand:* For networks that use certificate-based authentication. IT policies specify which domains require a VPN connection by using a VPN configuration profile.
- *Per App VPN:* For facilitating VPN connections on a much more granular basis. Mobile device management (MDM) solutions can specify a connection for each managed app and specific domains in Safari. This helps ensure that secure data always goes to and from the corporate network—and that a user's personal data doesn't.
- *Always On VPN:* Can be configured for devices managed through an MDM solution and supervised using Apple Configurator 2, Apple School Manager, or Apple Business Manager. Always On VPN eliminates the need for users to turn on VPN to enable protection when connecting to cellular and Wi-Fi networks. Always On VPN gives an organization full control over device traffic by tunneling all IP traffic back to the organization. The default exchange of parameters and keys for the subsequent encryption, IKEv2 secures traffic transmission with data encryption. The organization can monitor and filter traffic to and from its devices, secure data within its network, and restrict device access to the internet.

Wi-Fi security

Protocol security

Secure access to wireless networks

All Apple platforms support industry-standard Wi-Fi authentication and encryption protocols, to provide authenticated access and confidentiality when connecting to the following secure wireless networks:

- WPA2 Personal
- WPA2 Enterprise
- WPA2/WPA3 Transitional
- WPA3 Personal
- WPA3 Enterprise
- WPA3 Enterprise 192-bit security

WPA2 and WPA3 authenticate each connection and provide 128-bit AES encryption to ensure confidentiality for data sent over the air. This grants users the highest level of assurance that their data remains protected when sending and receiving communications over a Wi-Fi network connection.

WPA3 support

WPA3 is supported on the following Apple devices:

- iPhone 7 or later
- iPad 5th generation or later
- Apple TV 4K or later
- Apple Watch series 3 or later
- Mac computers (late 2013 on, with 802.11ac or later)

Newer devices support authentication with WPA3 Enterprise 192-bit security, including support for 256-bit AES encryption when connecting to compatible wireless access points (APs). This provides even stronger confidentiality protections for traffic sent over the air. WPA3 Enterprise 192-bit security is supported on iPhone 11, iPhone 11 Pro, iPhone 11 Pro Max, and later iOS and iPadOS devices.

PMF support

In addition to protecting data sent over the air, Apple platforms extend WPA2 and WPA3 level protections to unicast and multicast management frames through the Protected Management Frame (PMF) service defined in 802.11w. PMF support is available on the following Apple devices:

- iPhone 6 or later
- iPad Air 2 or later
- Apple TV HD or later
- Apple Watch series 3 or later
- Mac computers (late 2013 on, with 802.11ac or later)

With support for 802.1X, Apple devices can be integrated into a broad range of RADIUS authentication environments. 802.1X wireless authentication methods supported include EAP-TLS, EAP-TTLS, EAP-FAST, EAP-SIM, PEAPv0, and PEAPv1.

Platform protections

Apple operating systems protect the device from vulnerabilities in network processor firmware. This means that network controllers with Wi-Fi have limited access to Application Processor memory.

- When USB or SDIO (Secure Digital Input Output) is used to interface with the network processor, the network processor can't initiate direct memory access (DMA) transactions to the Application Processor.
- When PCIe is used, each network processor is on its own isolated PCIe bus. An Input/Output Memory Management Unit (IOMMU) on each PCIe bus further limits the network processor's DMA access to only memory and resources containing its network packets and control structures.

Deprecated protocols

Apple products support the following deprecated Wi-Fi authentication and encryption protocols:

- WEP Open, with both 40-bit and 104-bit keys
- WEP Shared, with both 40-bit and 104-bit keys
- Dynamic WEP
- Temporal Key Integrity Protocol (TKIP)
- WPA
- WPA/WPA2 Transitional

These protocols are no longer considered secure, and their use is strongly discouraged for compatibility, reliability, performance, and security reasons. They are supported for backward compatibility purposes only and may be removed in future software versions.

It's recommended that all Wi-Fi implementations be migrated to WPA3 Personal or WPA3 Enterprise, to provide the most robust, secure, and compatible Wi-Fi connections possible.

Wi-Fi privacy

MAC address randomization

Apple platforms use a randomized media access control address (MAC address) when performing Wi-Fi scans when not associated with a Wi-Fi network. These scans can be performed to find and connect to a known Wi-Fi network or to assist Location Services for apps that use geofences, such as location-based reminders or fixing a location in Apple Maps. Note that Wi-Fi scans that happen while trying to connect to a preferred Wi-Fi network aren't randomized. Wi-Fi MAC address randomization support is available on iPhone 5 or later.

Apple platforms also use a randomized MAC address when conducting enhanced Preferred Network Offload (ePNO) scans when a device isn't associated with a Wi-Fi network or its processor is asleep. ePNO scans are run when a device uses Location Services for apps that use geofences, such as location-based reminders that determine whether the device is near a specific location.

Because a device's MAC address changes when disconnected from a Wi-Fi network, it can't be used to persistently track a device by passive observers of Wi-Fi traffic, even when the device is connected to a cellular network. Apple has informed Wi-Fi manufacturers that iOS and iPadOS Wi-Fi scans use a randomized MAC address and that neither Apple nor manufacturers can predict these randomized MAC addresses.

iOS 14, iPadOS 14, and watchOS 7 introduce a new Wi-Fi privacy feature: When an iPhone, iPad, iPod touch, or Apple Watch connects to a Wi-Fi network, it identifies itself with a a unique (random) MAC address per network. This feature can be disabled either by the user or using a new option in the Wi-Fi payload. Under certain circumstances, the device will fall back to the actual MAC address.

For more information, see the Apple Support article Use private Wi-Fi addresses in iOS 14, iPadOS 14, and watchOS 7.

Wi-Fi frame sequence number randomization

Wi-Fi frames include a sequence number, which is used by the low-level 802.11 protocol to enable efficient and reliable Wi-Fi communications. Because these sequence numbers increment on each transmitted frame, they could be used to correlate information transmitted during Wi-Fi scans with other frames transmitted by the same device.

To guard against this, Apple devices randomize the sequence numbers whenever a MAC address is changed to a new randomized address. This includes randomizing the sequence numbers for each new scan request that's initiated while the device is unassociated. This randomization is supported on the following devices:

- iPhone 7 or later
- iPad 5th generation or later
- Apple TV 4K or later
- Apple Watch series 3 or later
- iMac Pro (Retina 5K, 27-inch, 2017) or later
- MacBook Pro (13-inch, 2018) or later
- MacBook Pro (15-inch, 2018) or later
- MacBook Air (Retina, 13-inch, 2018) or later
- Mac mini (2018) or later
- iMac (Retina 4K, 21.5-inch, 2019) or later
- iMac (Retina 5K, 27-inch, 2019) or later
- Mac Pro (2019) or later

Wi-Fi connections

Apple generates randomized MAC addresses for the Peer-to-Peer Wi-Fi connections that are used for AirDrop and AirPlay. Randomized addresses are also used for Personal Hotspot in iOS and iPadOS (with a SIM card) and Internet Sharing in macOS.

New random addresses are generated whenever these network interfaces are started, and unique addresses are independently generated for each interface as needed.

Hidden networks

Wi-Fi networks are identified by their network name, known as a *service set identifier (SSID)*. Some Wi-Fi networks are configured to hide their SSID, which results in the wireless access point not broadcasting the network's name. These are known as *hidden networks*. iPhone 6s or later automatically detects when a network is hidden. If a network is hidden, the iOS or iPadOS device sends a probe with the SSID included in the request—not otherwise. This prevents the device from broadcasting the name of previously hidden networks a user was connected to, thereby further ensuring privacy.

Bluetooth security

There are two types of Bluetooth in Apple devices, Bluetooth Classic and Bluetooth Low Energy (BLE). The Bluetooth security model for both versions includes the following distinct security features:

- *Pairing:* The process for creating one or more shared secret keys
- *Bonding:* The act of storing the keys created during pairing for use in subsequent connections to form a trusted device pair
- *Authentication:* Verifying that the two devices have the same keys
- *Encryption:* Message confidentiality
- *Message integrity:* Protection against message forgeries
- *Secure Simple Pairing:* Protection against passive eavesdropping and protection against man-in-the-middle attacks

Bluetooth version 4.1 added the Secure Connections feature to Bluetooth Classic (BR/EDR) physical transport.

The security features for each type of Bluetooth are listed below.

Support	Bluetooth Classic	Bluetooth Low Energy
Pairing	P-256 elliptic curve	FIPS-approved algorithms (AES-CMAC and P-256 elliptic curve)
Bonding	Pairing information stored in a secure location in iOS, iPad OS, macOS, tvOS, and watchOS devices	Pairing information stored in a secure location in iOS, iPadOS, macOS, tvOS, and watchOS devices
Authentication	FIPS-approved algorithms (HMAC-SHA256 and AES-CTR)	FIPS-approved algorithms
Encryption	AES-CCM cryptography, performed in the Controller	AES-CCM cryptography, performed in the Controller
Message integrity	AES-CCM, used for message integrity	AES-CCM, used for message integrity
Secure Simple Pairing: Protection against passive eavesdropping	Elliptic Curve Diffie-Hellman Exchange (ECDHE)	Elliptic Curve Diffie-Hellman Exchange (ECDHE)
Secure Simple Pairing: Protection against man-in-the-middle (MITM) attacks	Two user-assisted numeric methods: numerical comparison or passkey entry	Two user-assisted numeric methods: numerical comparison or passkey entry Pairings require a user response, including all non-MITM pairing modes
Bluetooth 4.1 or later	iMac Late 2015 or later MacBook Pro Early 2015 or later	iOS 9 or later iPadOS 13.1 or later macOS 10.12 or later tvOS 9 or later watchOS 2.0 or later

Support	Bluetooth Classic	Bluetooth Low Energy
Bluetooth 4.2 or later	iPhone 6 or later	iOS 9 or later iPadOS 13.1 or later macOS 10.12 or later tvOS 9 or later watchOS 2.0 or later

Bluetooth Low Energy privacy

To help secure user privacy, BLE includes the following two features: address randomization and cross-transport key derivation.

Address randomization is a feature that reduces the ability to track a BLE device over a period of time by changing the Bluetooth device address on a frequent basis. For a device using the privacy feature to reconnect to known devices, the device address, referred to as the *private address*, must be resolvable by the other device. The private address is generated using the device's identity resolving key exchanged during the pairing procedure.

iOS 13 or later and iPadOS 13.1 or later have the ability to derive link keys across transports, a feature known as *cross-transport key derivation*. For example, a link key generated with BLE can be used to derive a Bluetooth Classic link key. In addition, Apple added Bluetooth Classic to BLE support for devices that support the Secured Connections feature that was introduced in the Bluetooth Core Specification 4.1 (see the Bluetooth Core Specification 5.1).

Ultra Wideband security in iOS

The new Apple-designed U1 chip uses Ultra Wideband technology for spatial awareness—allowing iPhone 11, iPhone 11 Pro, and iPhone 11 Pro Max or later iPhone models to precisely locate other U1-equipped Apple devices. Ultra Wideband technology uses the same technology to randomize data found in other supported Apple devices:

- MAC address randomization
- Wi-Fi frame sequence number randomization

Single sign-on security

Single sign-on

iOS and iPadOS support authentication to enterprise networks through Single sign-on (SSO). SSO works with Kerberos-based networks to authenticate users to services they are authorized to access. SSO can be used for a range of network activities, from secure Safari sessions to third-party apps. Certificate-based authentication such as PKINIT is also supported.

macOS supports authentication to enterprise networks using Kerberos. Apps can use Kerberos to authenticate users to services they're authorized to access. Kerberos can also be used for a range of network activities, from secure Safari sessions and network file system authentication to third-party apps. Certificate-based authentication is supported, although app adoption of a developer API is required.

iOS, iPadOS, and macOS SSO use SPNEGO tokens and the HTTP Negotiate protocol to work with Kerberos-based authentication gateways and Windows Integrated Authentication systems that support Kerberos tickets. SSO support is based on the open source Heimdal project.

The following encryption types are supported in iOS, iPadOS, and macOS:

- AES-128-CTS-HMAC-SHA1-96
- AES-256-CTS-HMAC-SHA1-96
- DES3-CBC-SHA1
- ARCFOUR-HMAC-MD5

Safari supports SSO, and third-party apps that use standard iOS and iPadOS networking APIs can also be configured to use it. To configure SSO, iOS and iPadOS support a configuration profile payload that allows mobile device management (MDM) solutions to push down the necessary settings. This includes setting the user principal name (that is, the Active Directory user account) and Kerberos realm settings, as well as configuring which apps and Safari web URLs should be allowed to use SSO.

To configure Kerberos in macOS, acquire tickets with Ticket Viewer, log in to a Windows Active Directory domain, or use the `kinit` command-line tool.

Extensible Single sign-on

App developers can provide their own single sign-on implementations using SSO extensions. SSO extensions are invoked when a native or web app needs to use some identity provider for user authentication. Developers can provide two types of extensions: those that redirect to HTTPS and those that use a challenge/response mechanism such as Kerberos. This allows OpenID, OAuth, SAML2 and Kerberos authentication schemes to be supported by Extensible Single sign-on.

To use a Single sign-on extension, an app can either use the AuthenticationServices API or can rely on the URL interception mechanism offered by the operating system. WebKit and CFNetwork provide an interception layer that enables a seamless support of Single sign-on for any native or WebKit app. For a Single sign-on extension to be invoked, a configuration provided by an administrator has to be installed though a mobile device management (MDM) profile. In addition, redirect type extensions must use the Associated Domains payload to prove that the identity server they support is aware of their existence.

The only extension provided with the operating system is the Kerberos SSO extension.

AirDrop security

Apple devices that support AirDrop use Bluetooth Low Energy (BLE) and Apple-created peer-to-peer Wi-Fi technology to send files and information to nearby devices, including AirDrop-capable iOS devices running iOS 7 or later and Mac computers running OS X 10.11 or later. The Wi-Fi radio is used to communicate directly between devices without using any internet connection or wireless access point (AP). In macOS, this connection is encrypted with TLS.

AirDrop is set to share with Contacts Only by default. Users can also choose to use AirDrop to share with everyone, or turn off the feature entirely. Organizations can restrict the use of AirDrop for devices or apps being managed by using a mobile device management (MDM) solution.

AirDrop operation

AirDrop uses iCloud services to help users authenticate. When a user signs into iCloud, a 2048-bit RSA identity is stored on the device, and when the user enables AirDrop, an AirDrop short identity hash is created based on the email addresses and phone numbers associated with the user's Apple ID.

When a user chooses AirDrop as the method for sharing an item, the sending device emits an AirDrop signal over BLE that includes the user's AirDrop short identity hash. Other Apple devices that are awake, in close proximity, and have AirDrop turned on, detect the signal and respond using peer-to-peer Wi-Fi, so that the sending device can discover the identity of any responding devices.

In Contacts Only mode, the received AirDrop short identity hash is compared with hashes of people in the receiving device's Contacts app. If a match is found, the receiving device responds over peer-to-peer Wi-Fi with its identity information. If there is no match, the device doesn't respond.

In Everyone mode, the same overall process is used. However, the receiving device responds even if there is no match in the device's Contacts app.

The sending device then initiates an AirDrop connection using peer-to-peer Wi-Fi, using this connection to send a long identity hash to the receiving device. If the long identity hash matches the hash of a known person in the receiver's Contacts, then the receiver responds with its long identity hashes.

If the hashes are verified, the recipient's first name and photo (if present in Contacts) are displayed in the sender's AirDrop share sheet. In iOS and iPadOS, they are shown in the "People" or "Devices" section. Devices that aren't verified or authenticated are displayed in the sender's AirDrop share sheet with a silhouette icon and the device's name, as defined in Settings > General > About > Name. In iOS and iPadOS, they are placed in the "Other People" section of the AirDrop share sheet.

The sending user may then select whom they want to share with. Upon user selection, the sending device initiates an encrypted (TLS) connection with the receiving device, which exchanges their iCloud identity certificates. The identity in the certificates is verified against each user's Contacts app.

If the certificates are verified, the receiving user is asked to accept the incoming transfer from the identified user or device. If multiple recipients have been selected, this process is repeated for each destination.

Wi-Fi password sharing security on iPhone and iPad

iOS and iPadOS devices that support Wi-Fi password sharing use a mechanism similar to AirDrop to send a Wi-Fi password from one device to another.

When a user selects a Wi-Fi network (requestor) and is prompted for the Wi-Fi password, the Apple device starts a Bluetooth Low Energy (BLE) advertisement indicating that it wants the Wi-Fi password. Other Apple devices that are awake, in close proximity, and have the password for the selected Wi-Fi network connect using BLE to the requesting device.

The device that has the Wi-Fi password (grantor) requires the Contact information of the requestor, and the requestor must prove their identity using a similar mechanism to AirDrop. After identity is proven, the grantor sends the requestor the passcode which can be used to join the network.

Organizations can restrict the use of Wi-Fi password sharing for devices or apps being managed through a mobile device management (MDM) solution.

Firewall security in macOS

macOS includes a built-in firewall to protect the Mac from network access and denial-of-service attacks. It can be configured in the Security & Privacy pane of System Preferences and supports the following configurations:

- Block all incoming connections, regardless of app.
- Automatically allow built-in software to receive incoming connections.
- Automatically allow downloaded and signed software to receive incoming connections.
- Add or deny access based on user-specified apps.
- Prevent the Mac from responding to ICM (Internet Control Message Protocol) probing and portscan requests.

Developer kit security

Developer kit security overview

Apple provides a number of “kit” frameworks to enable third-party developers to extend Apple services. These frameworks are built with user security and privacy at their core:

- HomeKit
- CloudKit
- SiriKit
- DriverKit
- ReplayKit
- ARKit

HomeKit

HomeKit communication security

Overview

HomeKit provides a home automation infrastructure that uses iCloud and iOS, iPadOS, and macOS security to protect and sync private data without exposing it to Apple.

HomeKit identity and security are based on Ed25519 public-private key pairs. An Ed25519 key pair is generated on the iOS, iPadOS, and macOS device for each user for HomeKit, which becomes their HomeKit identity. It’s used to authenticate communication between iOS, iPadOS, and macOS devices, and between iOS, iPadOS, and macOS devices and accessories.

The keys—stored in keychain and are included only in encrypted Keychain backups—are kept up to date between devices using iCloud Keychain, where available. HomePod and Apple TV receive keys using tap-to-setup or the setup mode described below. Keys are shared from an iPhone to a paired Apple Watch using Apple Identity Service (IDS).

Communication between HomeKit accessories

HomeKit accessories generate their own Ed25519 key pair for use in communicating with iOS, iPadOS, and macOS devices. If the accessory is restored to factory settings, a new key pair is generated.

To establish a relationship between an iOS, iPadOS, and macOS device and a HomeKit accessory, keys are exchanged using Secure Remote Password (3072-bit) protocol utilizing an eight-digit code provided by the accessory's manufacturer, entered on the iOS, iPadOS device by the user, and then encrypted using ChaCha20-Poly1305 AEAD with HKDF-SHA512 derived keys. The accessory's MFi certification is also verified during setup. Accessories without an MFi chip can build in support for software authentication in iOS 11.3 or later.

When the iOS, iPadOS, and macOS device and the HomeKit accessory communicate during use, each authenticates the other using the keys exchanged in the above process. Each session is established using the Station-to-Station protocol and is encrypted with HKDF-SHA512 derived keys based on per-session Curve25519 keys. This applies to both IP-based and Bluetooth Low Energy (BLE) accessories.

For BLE devices that support broadcast notifications, the accessory is provisioned with a broadcast encryption key by a paired iOS, iPadOS, and macOS device over a secure session. This key is used to encrypt the data about state changes on the accessory, which are notified using the BLE advertisements. The broadcast encryption key is an HKDF-SHA512 derived key, and the data is encrypted using ChaCha20-Poly1305 AEAD algorithm. The broadcast encryption key is periodically changed by the iOS, iPadOS, and macOS device and updated to other devices using iCloud as described in HomeKit data security.

HomeKit and Siri

Siri can be used to query and control accessories, and to activate scenes. Minimal information about the configuration of the home is provided anonymously to Siri, to provide names of rooms, accessories, and scenes that are necessary for command recognition. Audio sent to Siri may denote specific accessories or commands, but such Siri data isn't associated with other Apple features such as HomeKit.

HomeKit data security

HomeKit data updates between devices and users

HomeKit data can be updated between a user's iOS, iPadOS, and macOS devices using iCloud and iCloud keychain. During this process, the HomeKit data is encrypted using keys derived from the user's HomeKit identity and a random nonce and is handled as an opaque binary large object, or *blob*. The most recent blob is stored in iCloud, but it isn't used for any other purpose. Because it's encrypted using keys that are available only on the user's iOS, iPadOS, and macOS devices, its contents are inaccessible during transmission and iCloud storage.

HomeKit data is also synced between multiple users of the same home. This process uses authentication and encryption that is the same as that used between an iOS, iPadOS, and macOS device and a HomeKit accessory. The authentication is based on Ed25519 public keys that are exchanged between the devices when a user is added to a home. After a new user is added to a home, all further communication is authenticated and encrypted using Station-to-Station protocol and per-session keys.

The user who initially created the home in HomeKit or another user with editing permissions can add new users. The owner's device configures the accessories with the public key of the new user so that the accessory can authenticate and accept commands from the new user. When a user with editing permissions adds a new user, the process is delegated to a home hub to complete the operation.

The process to provision Apple TV for use with HomeKit is performed automatically when the user signs in to iCloud. The iCloud account needs to have two-factor authentication enabled. Apple TV and the owner's device exchange temporary Ed25519 public keys over iCloud. When the owner's device and Apple TV are on the same local network, the temporary keys are used to secure a connection over the local network using Station-to-Station protocol and per-session keys. This process uses authentication and encryption that is the same as that used between an iOS, iPadOS, and macOS device and a HomeKit accessory. Over this secure local connection, the owner's device transfers the user's Ed25519 public-private key pairs to Apple TV. These keys are then used to secure the communication between Apple TV and the HomeKit accessories and also between Apple TV and other iOS, iPadOS, and macOS devices that are part of the HomeKit home.

If a user doesn't have multiple devices and doesn't grant additional users access to their home, no HomeKit data is transmitted to iCloud.

Home data and apps

Access to home data by apps is controlled by the user's Privacy settings. Users are asked to grant access when apps request home data, similar to Contacts, Photos, and other iOS, iPadOS, and macOS data sources. If the user approves, apps have access to the names of rooms, names of accessories, which room each accessory is in, and other information as detailed in the HomeKit developer documentation at https://developer.apple.com/homekit/.

Local data storage

HomeKit stores data about the homes, accessories, scenes, and users on a user's iOS, iPadOS, and macOS devices. This stored data is encrypted using keys derived from the user's HomeKit identity keys, plus a random nonce. Additionally, HomeKit data is stored using the Data Protection class Protected Until First User Authentication. HomeKit data is backed up only in encrypted backups, so, for example, unencrypted backups to iTunes (in macOS 10.14 or earlier) or the Finder (macOS 10.15 or later) through USB don't contain HomeKit data.

Securing routers with HomeKit

Routers that support HomeKit let users improve the security of their home network by managing the Wi-Fi access that HomeKit accessories have to their local network and to the internet. The routers also support Private PSK (PPSK) authentication, so accessories can be added to the Wi-Fi network using a key that's specific to the accessory and that can be revoked when needed. PPSK authentication improves security by not exposing the main Wi-Fi password to accessories, as well as by allowing the router to securely identify an accessory even if it were to change its MAC address.

Using the Home app, a user can configure access restrictions for groups of accessories as follows:

- *No restriction:* Allow unrestricted access to the internet and the local network.
- *Automatic:* This is the default setting. Allow access to the internet and the local network based on a list of internet sites and local ports provided to Apple by the accessory manufacturer. This list includes all sites and ports needed by the accessory in order to function properly. (No Restriction is in place until such a list is available.)
- *Restrict to Home:* No access to the internet or the local network except for the connections required by HomeKit to discover and control the accessory from the local network (including from the home hub to support remote control).

A PPSK is a strong, accessory-specific WPA2 Personal pass-phrase that is automatically generated by HomeKit and revoked if and when the accessory is later removed from the Home. A PPSK is used when an accessory is added to the Wi-Fi network by HomeKit in a Home that has been configured with a HomeKit router; this addition is reflected as Wi-Fi Credential: HomeKit-managed on the settings screen for the accessory in the Home app. Accessories that were added to the Wi-Fi network before adding the router are reconfigured to use a PPSK if the accessory supports this; otherwise, they retain their existing credentials.

As an additional security measure, users must configure the HomeKit router using the router manufacturer's app, so that the app can validate that users have access to the router and can add it to the Home app.

HomeKit camera security

Cameras that have an Internet Protocol address (IP address) in HomeKit send video and audio streams directly to the iOS, iPadOS, tvOS, and macOS device on the local network accessing the stream. The streams are encrypted using randomly generated keys on the device and an Internet Protocol camera (or IP camera), and they're exchanged over the secure HomeKit session to the camera. When a device isn't on the local network, the encrypted streams are relayed through the home hub to the device. The home hub doesn't decrypt the streams; it functions only as a relay between the device and the IP camera. When an app displays the HomeKit IP camera video view to the user, HomeKit renders the video frames securely from a separate system process. As a result, the app is unable to access or store the video stream. In addition, apps aren't permitted to capture screenshots from this stream.

HomeKit secure video

HomeKit provides an end-to-end secure and private mechanism to record, analyze, and view clips from HomeKit IP cameras without exposing that video content to Apple or any third party. When motion is detected by the IP camera, video clips are sent directly to an Apple device acting as a home hub, using a dedicated local network connection between that home hub and the IP camera. The local network connection is encrypted with a per-session HKDF-SHA512 derived key-pair that is negotiated over the HomeKit session between home hub and IP camera. HomeKit decrypts the audio and video streams on the home hub and analyzes the video frames locally for any significant event. If a significant event is detected, HomeKit encrypts the video clip using AES-256-GCM with a randomly generated AES256 key. HomeKit also generates poster frames for each clip and these poster frames are encrypted using the same AES256 key. The encrypted poster frame and audio and video data are uploaded to iCloud servers. The related metadata for each clip including the encryption key are uploaded to CloudKit using iCloud end-to-end encryption.

For face classification, HomeKit stores all data used to classify a particular person's face in CloudKit using iCloud end-to-end encryption. The data stored includes information about each person, such as name, as well as images representing that person's face. These face images can be sourced from a user's Photos if they opt in, or they can be collected from previously analyzed IP camera video. A HomeKit Secure Video analysis session uses this classification data to identify faces in the secure video stream it receives directly from the IP camera and includes that identification information in the clip metadata mentioned previously.

When the Home app is used to view the clips for a camera, the data is downloaded from iCloud and the keys to decrypt the streams are unwrapped locally using iCloud end-to-end decryption. The encrypted video content is streamed from the servers and decrypted locally on the iOS device before displaying it in the viewer. Each video clip session maybe broken down into sub-sections with each sub-section encrypting the content stream with its own unique key.

HomeKit security with Apple TV

Using third-party remote accessories with Apple TV

Some third-party remote accessories provide Human Interface Design (HID) events and Siri audio to an associated Apple TV added using the Home app. The remote sends the HID events over the secure session to the Apple TV. A Siri-capable TV remote sends audio data to Apple TV when the user explicitly activates the microphone on the remote using a dedicated Siri button. The remote sends the audio frames directly to the Apple TV using a dedicated local network connection. A per-session HKDF-SHA512 derived key-pair that is negotiated over the HomeKit session between Apple TV and the TV remote is used to encrypt the local network connection. HomeKit decrypts the audio frames on Apple TV and forwards them to the Siri app, where they are treated with the same privacy protections as all Siri audio input.

Apple TV profiles for HomeKit homes

When a user of a HomeKit home adds their profile to the owner of the home's Apple TV, it gives that user access to their TV shows, music, and podcasts. Settings for each user regarding their profile use on the Apple TV are shared to the owner's iCloud account using iCloud end-to-end encryption. The data is owned by each user and is shared as read-only to the owner. Each user of the home can change these values in the Home app and the Apple TV of the owner uses these settings.

When a setting is turned on, the iTunes account of the user is made available on the Apple TV. When a setting is turned off, all account and data pertaining to that user is deleted on the Apple TV. The initial CloudKit share is initiated by the user's device and the token to establish the secure CloudKit share is sent over the same secure channel that is used to sync data between users of the home.

HomeKit accessories and iCloud

Note: Whenever possible, access services directly through a home hub instead of through iCloud. For example, use a hub such as HomePod, Apple TV, or iPad.

iCloud remote access is still supported for legacy HomeKit devices. Apple carefully designed those devices so that users can control them and send notifications to them without revealing to Apple what the accessories are or what commands and notifications are being sent. HomeKit never sends information about the home over iCloud remote access.

Mutual authentication of an accessory and an Apple device

When a user sends a command using iCloud remote access, the accessory and iOS, iPadOS, and macOS device are mutually authenticated and data is encrypted using the same procedure described for local connections. The contents of the communications are encrypted and not visible to Apple. The addressing through iCloud is based on the iCloud identifiers registered during the setup process.

Accessory setup process

Accessories that support iCloud remote access are provisioned during the accessory's setup process. The provisioning process begins with the user signing in to iCloud. Next, the iOS, and iPadOS device asks the accessory to sign a challenge using the Apple Authentication Coprocessor that's built into all Built for HomeKit accessories. The accessory also generates prime256v1 elliptic curve keys, and the public key is sent to the iOS, and iPadOS device along with the signed challenge and the X.509 certificate of the authentication coprocessor. These are used to request a certificate for the accessory from the iCloud provisioning server. The certificate is stored by the accessory, but it doesn't contain any identifying information about the accessory, other than it has been granted access to HomeKit iCloud remote access. The iOS, and iPadOS device that is conducting the provisioning also sends a bag to the accessory, which contains the URLs and other information needed to connect to the iCloud remote access server. This information isn't specific to any user or accessory.

Accessory list of allowed users

Each accessory registers a list of allowed users with the iCloud remote access server. These users have been granted the ability to control the accessory by the user who added the accessory to the home. Users are granted an identifier by the iCloud server and can be mapped to an iCloud account for the purpose of delivering notification messages and responses from the accessories. Similarly, accessories have iCloud-issued identifiers, but these identifiers are opaque and don't reveal any information about the accessory itself.

How accessories connect to iCloud remote access server

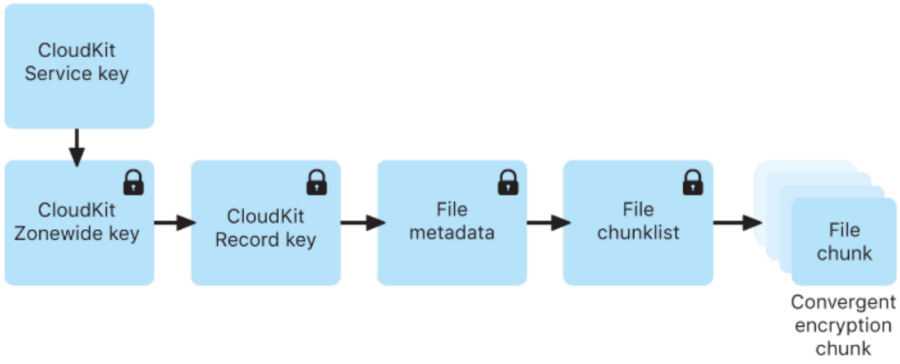
When an accessory connects to the HomeKit iCloud remote access server, it presents its certificate and a pass. The pass is obtained from a different iCloud server, and it isn't unique for each accessory. When an accessory requests a pass, it includes its manufacturer, model, and firmware version in its request. No user-identifying or home-identifying information is sent in this request. To help protect privacy, connection to the pass server isn't authenticated.

Accessories connect to the iCloud remote access server using HTTP/2, secured using TLS 1.2 with AES128-GCM and SHA256. The accessory keeps its connection to the iCloud remote access server open so that it can receive incoming messages and send responses and outgoing notifications to iOS, iPadOS, and macOS devices.

CloudKit security

CloudKit is a framework that lets app developers store key-value data, structured data, and assets in iCloud. Access to CloudKit is controlled using app entitlements. CloudKit supports both public and private databases. Public databases are used by all copies of the app, typically for general assets, and aren't encrypted. Private databases store the user's data.

As with iCloud Drive, CloudKit uses account-based keys to protect the information stored in the user's private database and, similar to other iCloud services, files are chunked, encrypted, and stored using third-party services. CloudKit uses a hierarchy of keys, similar to Data Protection. The per-file keys are wrapped by CloudKit Record keys. The Record keys, in turn, are protected by a zone-wide key, which is protected by the user's CloudKit Service key. The CloudKit Service key is stored in the user's iCloud account and is available only after the user has authenticated with iCloud.



CloudKit end-to-end encryption.

SiriKit security for iOS, iPadOS, and watchOS

Siri uses the app extension system to communicate with third-party apps. On a device, Siri can access the user's contact information and the device's current location. But before it provides protected data to an app, Siri checks the app's user-controlled access permissions. According to those permissions, Siri passes only the relevant fragment of the original user utterance to the app extension. For example, if an app doesn't have access to contact information, Siri won't resolve a relationship in a user request such as "Pay my mother 10 dollars using Payment App." In this case, the app would see only the literal term "my mother."

However, if the user has granted the app access to contact information, the app would receive resolved information about the user's mother. If a relationship is referenced in the body portion of a message—for example, "Tell my mother on MessageApp that my brother is awesome"—Siri doesn't resolve "my brother" regardless of the app's permissions.

SiriKit-enabled apps can send app-specific or user-specific vocabulary to Siri, such as the names of the user's contacts. This information allows Siri's speech recognition and natural language understanding to recognize vocabulary for that app and is associated with a random identifier. The custom information remains available as long as the identifier is in use, or until the user disables the app's Siri integration in Settings, or until the SiriKit-enabled app is uninstalled.

For an utterance like “Get me a ride to my mom’s home using RideShareApp,” the request requires location data from the user’s contacts. For that request only, Siri provides the required information to the app’s extension, regardless of the user permission settings for location or contact information for the app.

DriverKit security for macOS 10.15

DriverKit is the framework that allows developers to create device drivers that the user installs on their Mac. Drivers built with DriverKit run in user space, rather than as kernel extensions, for improved system security and stability. This makes for easier installation and increases the stability and security of macOS.

The user simply downloads the app (installers aren’t necessary when using system extensions or DriverKit) and the extension is enabled only when required. These replace kexts for many use cases, which require administrator privileges to install in /System/Library or /Library.

IT administrators who use device drivers, cloud storage solutions, networking, and security apps that require kernel extensions are encouraged to move to newer versions that are built on system extensions. These newer versions greatly reduce the possibility of kernel panics on the Mac as well as reduce the attack surface. These new extensions run in the user space, won’t require special privileges required for installation, and are automatically removed when the bundling app is moved to the Trash.

The DriverKit framework provides C++ classes for I/O services, device matching, memory descriptors, and dispatch queues. It also defines I/O-appropriate types for numbers, collections, strings, and other common types. The user uses these with family-specific driver frameworks like USBDriverKit and HIDDriverKit. Use the System Extensions framework to install and upgrade a driver.

ReplayKit security in iOS and iPadOS

ReplayKit is a beta framework that enables developers to add recording and live broadcasting capabilities to their apps. In addition, it allows users to annotate their recordings and broadcasts using the device’s front-facing camera and microphone.

Movie recording

There are several layers of security built into recording a movie:

- *Permissions dialog:* Before recording starts, ReplayKit presents a user consent alert requesting that the user acknowledge their intent to record the screen, the microphone, and the front-facing camera. This alert is presented once per app process, and it’s presented again if the app is left in the background for longer than 8 minutes.
- *Screen and audio capture:* Screen and audio capture occurs out of the app’s process in ReplayKit’s daemon replayd. This ensures the recorded content is never accessible to the app process.
- *In-app screen and audio capture:* This allows an app to get video and sample buffers, which is guarded by the permissions dialogue.

- *Movie creation and storage:* The movie file is written to a directory that's only accessible to ReplayKit's subsystems and is never accessible to any apps. This prevents recordings being used by third parties without the user's consent.
- *End-user preview and sharing:* The user has the ability to preview and share the movie with a user interface vended by ReplayKit. The user interface is presented out-of-process through the iOS Extension infrastructure and has access to the generated movie file.

ReplayKit broadcasting

There are several layers of security built into broadcasting a movie:

- *Screen and audio capture:* The screen and audio capture mechanism during broadcasting is identical to movie recording and occurs in `replayd`.
- *Broadcast extensions:* For third-party services to participate in ReplayKit broadcasting, they're required to create two new extensions that are configured with the com.apple.broadcast-services endpoint:
  - A user interface extension that allows the user to set up their broadcast
  - An upload extension that handles uploading video and audio data to the service's back-end servers

The architecture ensures that hosting apps have no privileges to the broadcasted video and audio contents. Only ReplayKit and the third-party broadcast extensions have access.

- *Broadcast picker:* With the broadcast picker, users initiate system broadcasts directly from their app using the same system-defined user interface that's accessible using Control Center. The user interface is implemented using a private API and is an extension that lives within the ReplayKit framework. It is out-of-process from the hosting app.
- *Upload extension:* The extension that third-party broadcast services implement to handle video and audio content during broadcasting uses raw unencoded sample buffers. During this mode of handling, video and audio data is serialized and passed to the third-party upload extension in real time through a direct XPC connection. Video data is encoded by extracting the IOSurface object from the video sample buffer, encoding it securely as an XPC object, sending it over through XPC to the third-party extension, and decoding it securely back into an IOSurface object.

ARKit security in iOS and iPadOS

ARKit is a framework that lets developers produce augmented reality experiences in their app or game. Developers can add 2D or 3D elements using the front or rear camera of an iOS or iPadOS device.

Apple designed cameras with privacy in mind, and third-party apps must obtain the user's consent before accessing the camera. In iOS and iPadOS, when a user grants an app access to their camera, that app can access real-time images from the front and rear cameras. Apps aren't allowed to use the camera without transparency that the camera is in use.

Photos and videos taken with the camera may contain other information, such as where and when they were taken, the depth of field, and overcapture. If users don't want photos and videos taken with the Camera app to include location, they can control this at any time by going to Settings > Privacy > Location Services > Camera. If users don't want photos and video to include location when shared, they can turn location off in the Options menu in the share sheet.

To better position the user's AR experience, apps that use ARKit can use world- or face-tracking information from the other camera. World tracking uses algorithms on the user's device to process information from these sensors to determine their position relative to a physical space. World tracking enables features such as Optical Heading in Maps.

Secure device management

Secure device management overview

iOS, iPadOS, macOS, and tvOS support flexible security policies and configurations that are easy to enforce and manage. Through them, organizations can protect corporate information and ensure that employees meet enterprise requirements, even if they are using devices they've provided themselves—for example, as part of a "bring your own device" (BYOD) program.

Organizations can use resources such as password protection, configuration profiles, remote wipe, and third-party mobile device management (MDM) solutions to manage fleets of devices and help keep corporate data secure, even when employees access this data on their personal devices.

In iOS 13 or later, iPadOS 13.1 or later, and macOS 10.15 or later, Apple devices support a new user enrollment option specifically designed for BYOD programs. User enrollments provide more autonomy for users on their own devices, while increasing the security of enterprise data by storing it on a separate, cryptographically protected APFS (Apple File System) volume. This provides a better balance of security, privacy, and user experience for BYOD programs.

Pairing model security for iPhone and iPad

iOS and iPadOS use a pairing model to control access to a device from a host computer. Pairing establishes a trust relationship between the device and its connected host, signified by public key exchange. iOS and iPadOS also use this sign of trust to enable additional functionality with the connected host, such as data syncing. In iOS 9 or later, services:

- That require pairing can't be started until after the device has been unlocked by the user
- Won't start unless the device has been recently unlocked
- May (such as with photo syncing) require the device to be unlocked to begin

The pairing process requires the user to unlock the device and accept the pairing request from the host. In iOS 9 or later, the user is also required to enter their passcode, after which the host and device exchange and save 2048-bit RSA public keys. The host is then given a 256-bit key that can unlock an escrow keybag stored on the device. The exchanged keys are used to start an encrypted SSL session, which the device requires before it sends protected data to the host or starts a service (iTunes or Finder syncing, file transfers, Xcode development, and so on). To use this encrypted session for all communication, the device requires connections from a host over Wi-Fi, so it must have been previously paired over USB. Pairing also enables several diagnostic capabilities. In iOS 9, if a pairing record hasn't been used for more than 6 months, it expires. In iOS 11 or later, this time frame is shortened to 30 days.

Certain diagnostic services, including com.apple.mobile.pcapd, are restricted to work only over USB. Additionally, the com.apple.file_relay service requires an Apple-signed configuration profile to be installed. In iOS 11 or later, Apple TV can use the Secure Remote Password protocol to wirelessly establish a pairing relationship.

A user can clear the list of trusted hosts with the Reset Network Settings or Reset Location & Privacy options.

Mobile device management

Mobile device management security overview

Overview

Apple operating systems support mobile device management (MDM), which allows organizations to securely configure and manage scaled Apple device deployments. MDM capabilities are built on existing operating system technologies, such as configuration profiles, over-the-air enrollment, and the Apple Push Notification service (APNs). For example, APNs is used to wake the device so it can communicate directly with its MDM solution over a secured connection. With APNs, no confidential or proprietary information is transmitted.

Using MDM, IT departments can enroll Apple devices in an enterprise environment, wirelessly configure and update settings, monitor compliance with corporate policies, manage software update policies, and even remotely wipe or lock managed devices.

In addition to the traditional device enrollments supported by iOS, iPadOS, macOS, and tvOS, an enrollment type has been added in iOS 13 or later, iPadOS 13.1 or later, and macOS 10.15 or later—User Enrollment. User enrollments are MDM enrollments specifically targeting "bring your own device" (BYOD) deployments where the device is personally owned but used in a managed environment. User enrollments grant the MDM solution more limited privileges than unsupervised device enrollments do, and provide cryptographic separation of user and corporate data.

Enrollment types

- *User Enrollment:* User Enrollment is designed for devices owned by the user and is integrated with Managed Apple IDs to establish a user identity on the device. Managed Apple IDs are part of the User Enrollment profile, and the user must successfully authenticate in order for enrollment to be completed. Managed Apple IDs can be used alongside a personal Apple ID that the user has already signed in with. Managed apps and accounts use a Managed Apple ID, and personal apps and accounts use a personal Apple ID.
- *Device Enrollment:* Device Enrollment allows organizations to have users manually enroll devices and then manage many different aspects of device use, including the ability to erase the device. Device Enrollment also has a larger set of payloads and restrictions that can be applied to the device. When a user removes an enrollment profile, all configuration profiles, their settings, and managed apps based on that enrollment profile are removed with it.
- *Automated Device Enrollment:* Automated Device Enrollment lets organizations configure and manage devices from the moment the devices are removed from the box (in a process known as *Auto Advance deployment*). These devices are known as *supervised*, and users have the option to prevent the MDM profile from being removed by the user. Automated Device Enrollment is designed for devices owned by the organization.

Device restrictions

Restrictions can be enabled—or in some cases, disabled—by administrators to prevent users from accessing a specific app, service, or function of an iPhone, iPad, Mac, or Apple TV that's enrolled in an MDM solution. Restrictions are sent to devices in a restrictions payload, which is part of a configuration profile. Certain restrictions on an iPhone may be mirrored on a paired Apple Watch.

Passcode and password settings management

By default, the user's passcode can be defined as a numeric PIN. In iOS and iPadOS devices with Touch ID or Face ID, the minimum passcode length is four digits. Because longer and more complex passcodes are harder to guess or attack, they are recommended.

Administrators can enforce complex passcode requirements and other policies using MDM or Microsoft Exchange ActiveSync, or by requiring users to manually install configuration profiles. An administrator password is needed for the macOS passcode policy payload installation. Some passcode policies can require a certain passcode length, composition, or other attributes.

Configuration profile enforcement

Configuration profiles are the primary way that an MDM solution delivers and manages policies and restrictions on managed devices. If organizations need to configure a large number of devices—or to provide lots of custom email settings, network settings, or certificates to a large number of devices—configuration profiles are a safe and secure way to do it.

Configuration profiles

A *configuration profile* is an XML file (ending in .mobileconfig) that consists of payloads that load settings and authorization information onto Apple devices. Configuration profiles automate the configuration of settings, accounts, restrictions, and credentials. These files can be created by an MDM solution or Apple Configurator 2, or they can be created manually. Before organizations send a configuration profile to an Apple device, they must enroll the device in the MDM solution using an enrollment profile.

Enrollment profiles

An *enrollment profile* is a configuration profile with an MDM payload that enrolls the device in the MDM solution specified for that device. This allows the MDM solution to send commands and configuration profiles to the device and to query certain aspects of the device. When a user removes an enrollment profile, all configuration profiles, their settings, and managed apps based on that enrollment profile are removed with it. There can be only one enrollment profile on a device at a time.

Configuration profile settings

A configuration profile contains a number of settings in specific payloads that can be specified, including (but not limited to):

- Passcode and password policies
- Restrictions on device features (for example, disabling the camera)
- Network and VPN settings
- Microsoft Exchange settings
- Mail settings
- Account settings
- LDAP directory service settings
- CalDAV calendar service settings
- Credentials and keys
- Software updates

Profile signing and encryption

Configuration profiles can be signed to validate their origin and encrypted to ensure their integrity and protect their contents. Configuration profiles for iOS and iPadOS are encrypted using the Cryptographic Message Syntax (CMS) specified in RFC 5652, supporting 3DES and AES128.

Profile installation

Users can install configuration profiles directly on their devices using Apple Configurator 2, or they can be downloaded using Safari, sent attached to a mail message, transferred using AirDrop or the Files app in iOS and iPadOS, or sent over the air using a mobile device management (MDM) solution. When a user sets up a device in Apple School Manager or Apple Business Manager, the device downloads and installs a profile for MDM enrollment. For information on how to remove profiles, see MDM Overview in MDM Settings for IT Administrators.

Note: On supervised devices, configuration profiles can also be locked to a device to completely prevent their removal or to allow removal only with a passcode. Because many organizations own their iOS and iPadOS devices, configuration profiles that bind a device to an MDM solution can be removed—but doing so also removes all managed configuration information, data, and apps.

Automated Device Enrollment

Organizations can automatically enroll iOS, iPadOS, macOS, and tvOS devices in mobile device management (MDM) without having to physically touch or prepare the devices before users get them. After enrolling in one of the services, administrators sign in to the service website and link the program to their MDM solution. The devices they purchased can then be assigned to users through MDM. During the device configuration process, security of sensitive data can be increased by ensuring appropriate security measures are in place. For example:

- Have users authenticate as part of the initial setup flow in the Apple device's Setup Assistant during activation.
- Provide a preliminary configuration with limited access and require additional device configuration to access sensitive data.

After a user has been assigned, any MDM-specified configurations, restrictions, or controls are automatically installed. All communications between devices and Apple servers are encrypted in transit through HTTPS (TLS).

The setup process for users can be further simplified by removing specific steps in the Setup Assistant for devices, so users are up and running quickly. Administrators can also control whether or not the user can remove the MDM profile from the device and ensure that device restrictions are in place throughout the lifecycle of the device. After the device is unboxed and activated, it can enroll in the organization's MDM solution—and all management settings, apps, and books are installed as defined by the MDM administrator.

Apple School Manager and Apple Business Manager

Apple School Manager and Apple Business Manager are services for IT administrators to deploy Apple devices that an organization has purchased directly from Apple or through participating Apple Authorized Resellers and carriers.

When used with an MDM solution, administrators can simplify the setup process for users, configure device settings, and distribute apps and books purchased in Apple School Manager and Apple Business Manager. Apple School Manager also integrates with Student Information Systems (SISs) directly or using SFTP, and Apple School Manager and Apple Business Manager can use System for Cross-domain Identity Management (SCIM) or federated authentication with Microsoft Azure Active Directory (Azure AD) so administrators can quickly create accounts.

Devices with iOS 11 or later and tvOS 10.2 or later can also be added to Apple School Manager and Apple Business Manager after the time of purchase using Apple Configurator 2.

Apple maintains certifications in compliance with the ISO/IEC 27001 and 27018 standards to enable Apple customers to address their regulatory and contractual obligations. These certifications provide our customers with an independent attestation over Apple's Information Security and Privacy practices for in-scope systems. For more information, see the Apple Support article Apple Internet Services Certifications.

Note: To learn whether an Apple program is available in a specific country or region, see the Apple Support article Availability of Apple programs for education and business.

Device supervision

Supervision generally denotes that the device is owned by the organization, giving them additional control over the device's configuration and restrictions.

iPhone and iPad devices with iOS 5 or later and Apple TV devices with tvOS 10.2 or later become supervised by:

- Using Apple Configurator 2 to supervise the device

  During this process, the device is erased and all data is lost.
- Enrolling the device in an MDM solution and selecting supervision as part of the enrollment process

Mac computers can be supervised if they:

- Are running macOS 11 enrolled in MDM using device enrollment
- Are upgraded to macOS 11 and the enrollment in MDM *was* a user approved MDM enrollment
- Are running macOS 10.14.4 or later and:
  - The devices' serial numbers appear in Apple School Manager or Apple Business Manager
  - Are enrolled in an MDM solution using Apple School Manager or Apple Business Manager

The following devices are supervised automatically when enrolled in Apple School Manager or Apple Business Manager:

- iPhone and iPod touch with iOS 13 or later
- iPad with iPadOS 13.1 or later
- Apple TV with tvOS 13 or later
- Mac computers with macOS 10.14.4 or later

Important: If the user knows the passcode, iPhone and iPad devices that aren't supervised can have manually installed configuration profiles removed, even if the option is set to "never." Manually installed configuration profiles for Mac computers can be removed using the `profiles` command-line tool, or System Preferences if the user knows an administrator's user name and password. As of macOS 10.15, like on iOS and iPadOS, profiles installed with MDM must be removed with MDM, or they are removed automatically upon unenrollment from MDM.

Activation Lock security

How Apple enforces Activation Lock varies depending on whether the device is an iPhone or an iPad, a Mac with Apple silicon, or an Intel-based Mac with the Apple T2 Security Chip.

Behavior on iPhone and iPad

On iPhone and iPad devices, Activation Lock is enforced through the activation process after the Wi-Fi selection screen in iOS and iPadOS Setup Assistant. When device indicates that its activating, it sends a request to an Apple server to get an activation certificate. Devices that are Activation Locked prompt the user for the iCloud credentials of the user that enabled Activation Lock at this time. iOS and iPadOS Setup Assistant won't progress unless a valid certificate can be obtained.

Behavior on a Mac with Apple silicon

In a Mac with Apple silicon, LLB verifies that a valid LocalPolicy for the device exists and that the LocalPolicy policy nonce values match the values stored in the Secure Storage Component. LLB boots to recoveryOS if:

- There is no LocalPolicy for the current macOS
- The LocalPolicy is invalid for that macOS
- The LocalPolicy nonce hash values don't match the hashes of values stored in the Secure Storage Component

recoveryOS detects that the Mac computer isn't activated and contacts the activation server to get an activation certificate. If the device is Activation Locked, recoveryOS prompts the user for iCloud credentials of the user that enabled Activation Lock at this time. After a valid activation certificate is obtained, that activation certificate key is used to obtain a RemotePolicy certificate. The Mac computer uses the LocalPolicy key and RemotePolicy certificate to produce a valid LocalPolicy. LLB won't allow booting of macOS unless a valid LocalPolicy is present.

Behavior on Intel-based Mac computers

In an Intel-based Mac with a T2 chip, the T2 chip firmware verifies that a valid activation certificate is present before allowing the computer to boot to macOS. UEFI firmware loaded by the T2 chip is responsible for querying the activation status of the device from the T2 chip and booting to recoveryOS instead of booting to macOS if a valid activation certificate isn't present. recoveryOS detects that the Mac isn't activated and contacts the activation server to get an activation certificate. If the device is Activation Locked, recoveryOS prompts the user for iCloud credentials of the user that enabled Activation Lock at this time. UEFI firmware won't allow booting of macOS unless a valid activation certificate is present.

Lost Mode, remote lock, and remote wipe

Managed Lost Mode is used to locate supervised devices when they are stolen. After they are located, they can be remotely locked or erased.

Lost Mode

If a supervised iOS or iPadOS device with iOS 9 or later is lost or stolen, an MDM administrator can remotely enable Lost Mode on that device. When Lost Mode is enabled, the current user is logged out and the device can't be unlocked. The screen displays a message that can be customized by the administrator, such as displaying a phone number to call if the device is found. When the device is put into Lost Mode, the administrator can request the device to send its current location (even if Location Services are off) and, optionally, play a sound. When an administrator turns off Lost Mode, which is the only way the mode can be exited, the user is informed of this action through a message on the Lock Screen or an alert on the Home screen.

Remote wipe, and remote lock

iOS, iPadOS, and macOS devices can be erased remotely by an administrator or user (instant remote wipe is available only if the Mac has FileVault enabled). Instant remote wipe is achieved by securely discarding the media key from Effaceable Storage, rendering all data unreadable. A remote wipe command can be initiated by mobile device management (MDM), Microsoft Exchange ActiveSync, or iCloud. On a Mac, the computer sends an acknowledgment and performs the wipe. With a remote lock, MDM requires that a six-digit passcode be applied to the Mac, rendering any user locked out until this passcode is typed in.

Note: Remote lock isn't available for a Mac with Apple silicon.

When a remote wipe command is triggered by MDM or iCloud, the device sends an acknowledgment and performs the wipe. For remote wipe through Microsoft Exchange ActiveSync, the device checks in with the Microsoft Exchange Server before performing the wipe. Remote wipe isn't possible in the following situations:

- With User Enrollment
- Using Microsoft Exchange ActiveSync when the account that was installed with User Enrollment
- Using Microsoft Exchange ActiveSync if the device is supervised

Users can also wipe iOS and iPadOS devices in their possession using the Settings app. And as mentioned, devices can be set to automatically wipe after a series of failed passcode attempts.

Shared iPad security in iPadOS

Overview

Shared iPad is a multiuser mode for use in iPad deployments. It allows users to share an iPad while maintaining separation of documents and data for each user. Each user gets their own private, reserved storage location, which is implemented as an APFS (Apple File System) volume protected by the user's credential. Shared iPad requires the use of a Managed Apple ID that's issued and owned by the organization and enables a user to sign in to any organizationally owned device that is configured for use by multiple users. User data is partitioned into separate directories, each in their own data protection domains and protected by both UNIX permissions and sandboxing. In iPadOS 13.4 or later, users can also sign in to a temporary session. When the user signs out of a temporary session, their APFS volume is deleted and its reserved space is returned to the system.

Signing in to Shared iPad

Both native and federated Managed Apple IDs are supported when signing in to Shared iPad. When using a federated account for the first time, the user is redirected to the Identity Provider's (IdP) sign-in portal. After authenticated, a short-lived access token is issued for the backing Managed Apple IDs—and the login process proceeds similarly to the native Managed Apple IDs sign-in process. Once signed in, Setup Assistant on Shared iPad prompts the user to establish a passcode (credential) used to secure the local data on the device and to authenticate to the login screen in the future. Like a single-user device, in which the user would sign in once to their Managed Apple ID using their federated account and then unlock their device with their passcode, on Shared iPad the user signs in once using their federated account and from then on uses their established passcode.

When a user signs in without federated authentication, the Managed Apple ID is authenticated with Apple Identity Service (IDS) using the SRP protocol. If authentication is successful, a short-lived access token specific to the device is granted. If the user has used the device before, they already have a local user account, which is unlocked using the same credential.

If the user hasn't used the device before or is using the temporary session feature, Shared iPad provisions a new UNIX user ID, an APFS volume to store the user's personal data, and a local keychain. Because storage is allocated (reserved) for the user at the time the APFS volume is created, there may be insufficient space to create a new volume. In such an event, the system will identify an existing user whose data has finished syncing to the cloud and evict that user from the device in order to allow the new user to sign in. In the unlikely event that all existing users haven't completed uploading their cloud data, the new user sign in fails. To sign in, the new user will need to wait for one user's data to finish syncing, or have an administrator forcibly delete an existing user account, thereby risking data loss.

If the device isn't connected to the internet (for example, if the user has no Wi-Fi access point), authentication can occur against the local account for a limited number of days. In that situation, only users with previously existing local accounts or a temporary session can sign in. After the time limit has expired, users are required to authenticate online, even if a local account already exists.

After a user's local account has been unlocked or created, if it's remotely authenticated, the short-lived token issued by Apple's servers is converted to an iCloud token that permits signing in to iCloud. Next, the users' settings are restored and their documents and data are synced from iCloud.

While a user session is active and the device remains online, documents and data are stored on iCloud as they are created or modified. In addition, a background syncing mechanism ensures that changes are pushed to iCloud, or to other web services using NSURLSession background sessions, after the user signs out. After background syncing for that user is complete, the user's APFS volume is unmounted and can't be mounted again without the user signing back in.

Temporary sessions don't sync data with iCloud, and although a temporary session can sign into a third-party syncing service such as Box or Google Drive, there's no facility to continue syncing data when the temporary session ends.

Signing out of Shared iPad

When a user signs out of Shared iPad, that user's keybag is immediately locked and all apps are shut down. To accelerate the case of a new user signing in, iPadOS defers some ordinary sign-out actions temporarily and presents a login window to the new user. If a user signs in during this time (approximately 30 seconds), Shared iPad performs the deferred cleanup as part of signing in to the new user account. However, if Shared iPad remains idle, it triggers the deferred cleanup. During the cleanup phase, Login Window is restarted as if another sign-out had occurred.

When a temporary session is ended, Shared iPad performs the full logout sequence and deletes the temporary session's APFS volume immediately.

Apple Configurator 2 security

Apple Configurator 2 features a flexible, secure, device-centric design that enables an administrator to quickly and easily configure one or dozens of iOS, iPadOS, and tvOS devices connected to a Mac through USB (or tvOS devices paired through Bonjour) before giving them to users. With Apple Configurator 2, an administrator can update software, install apps and configuration profiles, rename and change wallpaper on devices, export device information and documents, and much more.

Administrators can also choose to add iOS, iPadOS, and tvOS devices to Apple School Manager or Apple Business Manager using Apple Configurator 2, even if the devices weren't purchased directly from Apple, an Apple Authorized Reseller, or an authorized cellular carrier. When the administrator sets up a device that has been manually enrolled, it behaves like any other enrolled device, with mandatory supervision and mobile device management (MDM) enrollment. For devices that weren't purchased directly, the user has a 30-day provisional period to remove the device from enrollment, supervision, and MDM. The 30-day provisional period begins after the device is activated.

If iOS, iPadOS, and tvOS devices that have absolutely no internet connection and that are connected to a host Mac with an internet connection while the devices are being set up, organizations can use Apple Configurator 2 to activate them. Administrators can restore, activate, and prepare devices with their necessary configuration including Apps, Profiles, and Documents without ever needing to connect to either Wi-Fi or cellular networks. This feature doesn't allow an administrator to bypass any existing Activation Lock requirements normally required during nontethered activation.

Screen Time security

Screen Time is a feature—in iOS 12 or later, iPadOS, and macOS 10.15 or later, and some features of watchOS 6 or later—that lets users understand and control their own app and web usage, or that of their children. Although Screen Time isn't a new system security feature, it's important to understand how Screen Time protects the security and privacy of the data gathered and shared between devices.

In Screen Time, there are two types of users: adults and children.

The table below describes the main features of Screen Time.

Feature	Supported operating system
View usage data	iOS iPadOS macOS
Enforce additional restrictions	iOS iPadOS macOS watchOS
Set web usage limits	iOS iPadOS macOS
Set app limits	iOS iPadOS macOS watchOS
Configure Downtime	iOS iPadOS macOS watchOS

For users managing their own device usage, Screen Time controls and usage data can be synced across devices associated to the same iCloud account using CloudKit end-to-end encryption. This requires that the user's account have two-factor authentication enabled (syncing is on by default). Screen Time replaces the Restrictions feature found in previous versions of iOS and iPadOS, and the Parental Controls feature found in previous versions of macOS.

In iOS 13 or later, iPadOS 13.1 or later, and macOS 10.15 or later, Screen Time users and managed children automatically share their usage across devices if their iCloud account has two-factor authentication enabled. When a user clears Safari history or deletes an app, the corresponding usage data is removed from the device and all synced devices.

Parents and Screen Time

Parents can also use Screen Time in iOS, iPadOS, and macOS devices to understand and control their children's use. If the parent is a family organizer (in iCloud Family Sharing), they can view usage data and manage Screen Time settings for their children. Children are informed when their parents turn on Screen Time, and they can monitor their own usage as well. When parents turn on Screen Time for their children, the parents set a passcode so their children can't make changes. When they reach age of majority (age will vary depending on country or region), the children can turn this monitoring off.

Usage data and configuration settings are transferred between the parent's and child's devices using the end-to-end encrypted Apple Identity Service (IDS) protocol. Encrypted data may be briefly stored on IDS servers until it's read by the receiving device (for example, as soon as the iPhone, iPad, or iPod touch is turned on, if it was off). This data isn't readable by Apple.

Screen Time analytics

If the user turns on Share iPhone & Watch Analytics, only the following anonymized data is collected so that Apple can better understand how Screen Time is being used:

- Was Screen Time turned on during Setup Assistant or later in Settings
- Change in Category usage after creating a limit for it (within 90 days)
- Is Screen Time turned on
- Is Downtime enabled
- Number of times the "Ask for more" query was used
- Number of app limits
- Number of times users viewed usage in the Screen Time settings, per user type and per view type (local, remote, widget)
- Number of times users ignore a limit, per user type
- Number of times users delete a limit, per user type

No specific app or web usage data is gathered by Apple. When a user sees a list of apps in Screen Time usage information, the app icons are pulled directly from the App Store, which doesn't retain any data from these requests.

Glossary

Address Space Layout Randomization (ASLR) A technique employed by operating systems to make the successful exploitation by a software bug much more difficult. By ensuring memory addresses and offsets are unpredictable, exploit code can't hard code these values.

AES (Advanced Encryption Standard) A popular global encryption standard used to encrypt data to keep it private.

AES cryptographic engine A dedicated hardware component that implements AES.

AES-XTS A mode of AES defined in IEEE 1619-2007 meant to work for encrypting storage media.

APFS (Apple File System) The default file system for iOS, iPadOS, tvOS, watchOS, and Mac computers using macOS 10.13 or later. APFS features strong encryption, space sharing, snapshots, fast directory sizing, and improved file system fundamentals.

Apple Business Manager Apple Business Manager is a simple, web-based portal for IT administrators that provides a fast, streamlined way for organizations to deploy Apple devices that they have purchased directly from Apple or from a participating Apple Authorized Reseller or carrier. They can automatically enroll devices in their mobile device management (MDM) solution without having to physically touch or prepare the devices before users get them.

Apple Identity Service (IDS) Apple's directory of iMessage public keys, APNs addresses, and phone numbers and email addresses that are used to look up the keys and device addresses.

Apple Push Notification service (APNs) A worldwide service provided by Apple that delivers push notifications to Apple devices.

Apple School Manager Apple School Manager is a simple, web-based portal for IT administrators that provides a fast, streamlined way for organizations to deploy Apple devices that they have purchased directly from Apple or from a participating Apple Authorized Reseller or carrier. They can automatically enroll devices in their mobile device management (MDM) solution without having to physically touch or prepare the devices before users get them.

Apple Security Bounty A reward given by Apple to researchers who report a vulnerability that affects the latest shipping operating systems and, where relevant, the latest hardware.

Boot Camp Boot Camp supports the installation of Microsoft Windows on supported Mac computers.

Boot Progress Register (BPR) A set of system on chip (SoC) hardware flags that software can use to track the boot modes the device has entered, such as Device Firmware Update (DFU) mode and Recovery mode. After a Boot Progress Register flag is set, it can't be cleared. This allows later software to get a trusted indicator of the state of the system.

Boot ROM The very first code executed by a device's processor when it first boots. As an integral part of the processor, it can't be altered by either Apple or an attacker.

CKRecord A dictionary of key-value pairs that contain data saved to or fetched from CloudKit.

Data Protection File and Keychain protection mechanism for supported Apple devices. It can also refer to the APIs that apps use to protect files and keychain items.

Data Vault A mechanism—enforced by the kernel—to protect against unauthorized access to data regardless of whether the requesting app is itself sandboxed.

Device Firmware Upgrade (DFU) mode A mode in which a device's Boot ROM code waits to be recovered over USB. The screen is black when in DFU mode, but upon connecting to a computer running iTunes or the Finder, the following prompt is presented: "iTunes (or the Finder) has detected an (iPad, iPhone, or iPod touch) in Recovery mode. The user must restore this (iPad, iPhone, or iPod touch) before it can be used with iTunes (or the Finder)."

direct memory access (DMA) A feature that enables hardware subsystems to access main memory independent of the CPU.

Effaceable Storage A dedicated area of NAND storage, used to store cryptographic keys, that can be addressed directly and wiped securely. While it doesn't provide protection if an attacker has physical possession of a device, keys held in Effaceable Storage can be used as part of a key hierarchy to facilitate fast wipe and forward security.

Elliptic Curve Diffie-Hellman Exchange (ECDHE) Elliptic Curve Diffie-Hellman Exchange with ephemeral keys. ECDHE allows two parties to agree on a secret key in a way that prevents the key from being discovered by an eavesdropper watching the messages between the two parties.

Elliptic Curve Digital Signature Algorithm (ECDSA) Elliptic Curve Digital Signature Algorithm (ECDSA) is a digital signature algorithm based on elliptic curve cryptography.

eSPI The Enhanced Serial Peripheral Interface bus for synchronous serial communication.

Exclusive Chip Identification (ECID) A 64-bit identifier that's unique to the processor in each iOS and iPadOS device. When a call is answered on one device, ringing of nearby iCloud-paired devices is terminated by briefly advertising through Bluetooth Low Energy (BLE) 4.0. The advertising bytes are encrypted using the same method as Handoff advertisements. Used as part of the personalization process, it's not considered a secret.

file system key The key that encrypts each file's metadata, including its class key. This is kept in Effaceable Storage to facilitate fast wipe, rather than confidentiality.

group ID (GID) Like the UID, but common to every processor in a class.

hardware security module (HSM) A specialized tamper-resistant computer that safeguards and manages digital keys.

HMAC A hash-based message authentication code based on a cryptographic hash function.

iBoot Code that loads XNU, as part of the secure boot chain. Depending on the system on chip (SoC) generation, iBoot may be loaded by the Low-Level Bootloader or directly by the Boot ROM.

Input/Output Memory Management Unit (IOMMU) An input/output memory management unit. A subsystem in an integrated chip that controls access to address space from other input/output devices and peripherals.

integrated circuit (IC) Also known as a *microchip*.

Joint Test Action Group (JTAG) A standard hardware debugging tool used by programmers and circuit developers.

keybag A data structure used to store a collection of class keys. Each type (user, device, system, backup, escrow, or iCloud Backup) has the same format.

A header containing: Version (set to four in iOS 12 or later), Type (system, backup, escrow, or iCloud Backup), Keybag UUID, an HMAC if the keybag is signed, and the method used for wrapping the class keys—tangling with the UID or PBKDF2, along with the salt and iteration count.

A list of class keys: Key UUID, Class (which file or Keychain Data Protection class), wrapping type (UID-derived key only; UID-derived key and passcode-derived key), wrapped class key, and a public key for asymmetric classes.

keychain The infrastructure and a set of APIs used by Apple operating systems and third-party apps to store and retrieve passwords, keys, and other sensitive credentials.

key wrapping Encrypting one key with another. iOS and iPadOS use NIST AES key wrapping, in accordance with RFC 3394.

Low-Level Bootloader (LLB) On Mac computers with a two-stage boot architecture, LLB contains the code that's invoked by the Boot ROM and that in turn loads iBoot, as part of the secure boot chain.

media key Part of the encryption key hierarchy that helps provide for a secure and instant wipe. In iOS, iPadOS, tvOS, and watchOS, the media key wraps the metadata on the data volume (and thus without it access to all per-file keys is impossible, rendering files protected with Data Protection inaccessible). In macOS, the media key wraps the keying material, all metadata, and data on the FileVault protected volume. In either case, wipe of the media key renders encrypted data inaccessible.

memory controller The subsystem in a system on chip that controls the interface between the system on chip and its main memory.

mobile device management (MDM) A service that lets the user remotely manage enrolled devices. After a device is enrolled, the user can use the MDM service over the network to configure settings and perform other tasks on the device without user interaction.

NAND Nonvolatile flash memory.

per-file key The key used by Data Protection to encrypt a file on the file system. The per-file key is wrapped by a class key and is stored in the file's metadata.

provisioning profile A property list (.plist file) signed by Apple that contains a set of entities and entitlements allowing apps to be installed and tested on an iOS or iPadOS device. A development provisioning profile lists the devices that a developer has chosen for ad hoc distribution, and a distribution provisioning profile contains the app ID of an enterprise-developed app.

Recovery mode A mode used to restore many Apple devices if it doesn't recognize the user's device so the user can reinstall the operating system.

ridge flow angle mapping A mathematical representation of the direction and width of the ridges extracted from a portion of a fingerprint.

Secure Storage Component On supported devices, the Secure Enclave is paired with a Secure Storage Component for anti-replay nonce storage. A chip designed with immutable RO code, a hardware random number generator, cryptography engines, and physical tamper detection. To read and update nonces, the Secure Enclave and storage chip employ a secure protocol that ensures exclusive access to the nonces. There are multiple generations of this technology with differing security guarantees.

sepOS The Secure Enclave firmware, based on an Apple-customized version of the L4 microkernel.

software seed bits Dedicated bits in the Secure Enclave AES Engine that get appended to the UID when generating keys from the UID. Each software seed bit has a corresponding lock bit. The Secure Enclave Boot ROM and operating system can independently change the value of each software seed bit as long as the corresponding lock bit hasn't been set. After the lock bit is set, neither the software seed bit nor the lock bit can be modified. The software seed bits and their locks are reset when the Secure Enclave reboots.

SSD controller A hardware subsystem that manages the storage media (solid-state drive).

System Coprocessor Integrity Protection (SCIP) A mechanism Apple uses to prevent modification of coprocessor firmware.

system on chip (SoC) An integrated circuit (IC) that incorporates multiple components into a single chip. The Application Processor, the Secure Enclave, and other coprocessors are components of the SoC.

system software authorization A process that combines cryptographic keys built into hardware with an online service to check that only legitimate software from Apple, appropriate to supported devices, is supplied and installed at upgrade time.

tangling The process by which a user's passcode is turned into a cryptographic key and strengthened with the device's UID. This process ensures that a brute-force attack must be performed on a given device, and thus is rate limited and can't be performed in parallel. The tangling algorithm is PBKDF2, which uses AES keyed with the device UID as the pseudorandom function (PRF) for each iteration.

UEFI firmware Unified Extensible Firmware Interface, a replacement technology for BIOS to connect firmware to a computer's operating system.

Uniform Resource Identifier (URI) A string of characters that identifies a web-based resource.

unique ID (UID) A 256-bit AES key that's burned into each processor at manufacture. It can't be read by firmware or software, and it's used only by the processor's hardware AES Engine. To obtain the actual key, an attacker would have to mount a highly sophisticated and expensive physical attack against the processor's silicon. The UID isn't related to any other identifier on the device including, but not limited to, the UDID.

xART eXtended Anti-Replay Technology or a set of services that provides encrypted, authenticated persistent storage for the Secure Enclave with anti-replay capabilities based on the physical storage architecture. See Secure Storage Component.

XNU The kernel at the heart of the Apple operating systems. It's assumed to be trusted, and it enforces security measures such as code signing, sandboxing, entitlement checking, and Address Space Layout Randomization (ASLR).

Document revision history

Date	Summary
February 2021	Updated for: • iOS 14.3 • iPadOS 14.3 • macOS 11.1 • tvOS 14.3 • watchOS 7.2 Topics added: • Memory safe iBoot implementation • Boot process for a Mac with Apple silicon • Boot modes for a Mac with Apple silicon • Startup Disk security policy control for a Mac with Apple silicon • LocalPolicy signing-key creation and management • Contents of a LocalPolicy file for a Mac with Apple silicon • Signed system volume security in macOS • Apple Security Research Device • Password Monitoring • IPv6 security • Car keys security in iOS Topics updated: • Secure Enclave • Hardware microphone disconnect • recoveryOS and diagnostics environments for an Intel-based Mac • Direct memory access protections for Mac computers • Kernel extensions in macOS • System Integrity Protection • System security for watchOS • Managing FileVault in macOS • App access to saved passwords • Password security recommendations • Apple Cash security in iOS, iPadOS, and watchOS • Secure Business Chat using the Messages app • Wi-Fi privacy • Activation Lock security • Apple Configurator 2 security

Date	Summary
April 2020	Updated for: • iOS 13.4 • iPadOS 13.4 • macOS 10.15.4 • tvOS 13.4 • watchOS 6.2 Updates: • iPad microphone disconnect added to Hardware microphone disconnect. • Data vaults added to Protecting app access to user data. • Updates to Managing FileVault in macOS and Command-line tools. • Malware Removal Tool additions in Protecting against malware in macOS. • Updates to Shared iPad security in iPadOS.
December 2019	Merged the iOS Security Guide, macOS Security Overview, and the Apple T2 Security Chip Overview Updated for: • iOS 13.3 • iPadOS 13.3 • macOS 10.15.2 • tvOS 13.3 • watchOS 6.1.1 Privacy Controls, Siri and Siri Suggestions, and Safari Intelligent Tracking Prevention have been removed. See https://www.apple.com/privacy/ for the latest on those features.
May 2019	Updated for iOS 12.3 • Support for TLS 1.3 • Revised description of AirDrop security • DFU mode and Recovery mode • Passcode requirements for accessory connections
November 2018	Updated for iOS 12.1 • Group FaceTime
September 2018	Updated for iOS 12 • Secure Enclave • OS Integrity Protection • Express Card with power reserve • DFU mode and Recovery mode • HomeKit TV Remote accessories • Contactless passes • Student ID cards • Siri Suggestions • Shortcuts in Siri • Shortcuts app • User password management • Screen Time • Security Certifications and programs

Date	Summary
July 2018	Updated for iOS 11.4 • Biometric policies • HomeKit • Apple Pay • Business Chat • Messages in iCloud • Apple Business Manager
December 2017	Updated for iOS 11.2 • Apple Pay Cash
October 2017	Updated for iOS 11.1 • Security Certifications and programs • Touch ID/Face ID • Shared Notes • CloudKit end-to-end encryption • TLS update • Apple Pay, Paying with Apple Pay on the web • Siri Suggestions • Shared iPad
July 2017	Updated for iOS 10.3 • Secure Enclave • File Data Protection • Keybags • Security Certifications and programs • SiriKit • HealthKit • Network Security • Bluetooth • Shared iPad • Lost Mode • Activation Lock • Privacy Controls
March 2017	Updated for iOS 10 • System Security • Data Protection classes • Security Certifications and programs • HomeKit, ReplayKit, SiriKit • Apple Watch • Wi-Fi, VPN • Single sign-on • Apple Pay, Paying with Apple Pay on the web • Credit, debit, and prepaid card provisioning • Safari Suggestions

Date	Summary
May 2016	Updated for iOS 9.3 • Managed Apple ID • Two-factor authentication for Apple ID • Keybags • Security Certifications • Lost Mode, Activation Lock • Secure Notes • Apple School Manager • Shared iPad
September 2015	Updated for iOS 9 • Apple Watch Activation Lock • Passcode policies • Touch ID API support • Data Protection on A8 uses AES-XTS • Keybags for unattended software update • Certification updates • Enterprise app trust model • Data Protection for Safari bookmarks • App Transport Security • VPN specifications • iCloud Remote Access for HomeKit • Apple Pay Rewards cards, Apple Pay card issuer's app • Spotlight on-device indexing • iOS Pairing Model • Apple Configurator 2 • Restrictions

Apple
One Apple Park Way
Cupertino, CA 95014
USA
apple.com

028-00309