Apple Platform Security

Fall 2019

Introduction to Apple platform security

Apple designs security into the core of its platforms. Building on the experience of creating the world's most advanced mobile operating system, Apple has created security architectures that address the unique requirements of mobile, watch, desktop, and home.

Every Apple device combines hardware, software, and services designed to work together for maximum security and a transparent user experience in service of the ultimate goal of keeping personal information safe. Custom security hardware powers critical security features. Software protections work to keep the operating system and third-party apps safe. Services provide a mechanism for secure and timely software updates, power a safer app ecosystem, secure communications and payments, and provide a safer experience on the Internet. Apple devices protect not only the device and its data, but the entire ecosystem, including everything users do locally, on networks, and with key Internet services.

Just as we design our products to be simple, intuitive, and capable, we design them to be secure. Key security features, such as hardware-based device encryption, can't be disabled by mistake. Other features, such as Touch ID and Face ID, enhance the user experience by making it simpler and more intuitive to secure the device. And because many of these features are enabled by default, users or IT departments don't need to perform extensive configurations.

This documentation provides details about how security technology and features are implemented within Apple platforms. It also helps organizations combine Apple platform security technology and features with their own policies and procedures to meet their specific security needs.

The content is organized into the following topic areas:

- **Hardware Security and Biometrics:** The hardware that forms the foundation for security on Apple devices, including the Secure Enclave, a dedicated AES crypto engine, Touch ID, and Face ID.
- **System Security:** The integrated hardware and software functions that provide for the safe boot, update, and ongoing operation of Apple operating systems.
- **Encryption and Data Protection:** The architecture and design that protects user data if the device is lost or stolen, or if an unauthorized person attempts to use or modify it.
- **App Security:** The software and services that provide a safe app ecosystem and enable apps to run securely and without compromising platform integrity.
- **Services Security:** Apple's services for identification, password management, payments, communications, and finding lost devices.
- **Network Security:** Industry-standard networking protocols that provide secure authentication and encryption of data in transmission.

- **Developer Kits:** Frameworks for secure and private management of home and health, as well as extension of Apple device and service capabilities to third-party apps.
- **Secure Device Management:** Methods that allow management of Apple devices, prevent unauthorized use, and enable remote wipe if a device is lost or stolen.
- **Security Certifications and Programs:** Information on ISO certifications, Cryptographic validation, Common Criteria Certification, and the Commercial Solutions for Classified (CSfC) Program.

A commitment to security

Apple is committed to helping protect customers with leading privacy and security technologies—designed to safeguard personal information—and comprehensive methods—to help protect corporate data in an enterprise environment. Apple rewards researchers for the work they do to uncover vulnerabilities by offering the Apple Security Bounty. Details of the program and bounty categories are available at https://developer.apple.com/security-bounty/.

We maintain a dedicated security team to support all Apple products. The team provides security auditing and testing for products, both under development and released. The Apple team also provides security tools and training, and actively monitors for threats and reports of new security issues. Apple is a member of the Forum of Incident Response and Security Teams (FIRST).

Apple continues to push the boundaries of what is possible in security and privacy. For example, Find My uses existing cryptographic primitives to enable the groundbreaking capability of distributed finding of an offline Mac — without exposing to anyone, including Apple, the identity or location data of any of the users involved. To enhance Mac firmware security, Apple has leveraged an analog to page tables to block inappropriate access from peripherals, but at a point so early in the boot process that RAM hasn't yet been loaded. And as attackers continue to increase the sophistication of their exploit techniques, Apple is dynamically controlling memory execution privileges for iPhone and iPad by leveraging custom CPU instructions — unavailable on any other mobile devices — to thwart compromise. Just as important as the innovation of new security capabilities, new features are built with privacy and security at their center of their design.

To make the most of the extensive security features built into our platforms, organizations are encouraged to review their IT and security policies to ensure that they are taking full advantage of the layers of security technology offered by these platforms.

To learn more about reporting issues to Apple and subscribing to security notifications, see Report a security or privacy vulnerability.

Apple believes privacy is a fundamental human right and has numerous built-in controls and options that allow users to decide how and when apps use their information, as well as what information is being used.To learn more about Apple's approach to privacy, privacy controls on Apple devices, and the Apple privacy policy, see https://www.apple.com/privacy.

Note: Unless otherwise noted, this documentation covers the following operating system versions: iOS 13.3, iPadOS 13.3, macOS 10.15.2, tvOS 13.3, and watchOS 6.1.1.

Hardware Security and Biometrics

Hardware security overview

Secure software requires a foundation of security built into hardware. That's why Apple devices—running iOS, iPadOS, macOS, watchOS, or tvOS—have security capabilities designed into silicon. These include custom CPU capabilities that power system security features and silicon dedicated to security functions. The most critical component is the Secure Enclave coprocessor, which appears on all modern iOS, iPadOS, watchOS, and tvOS devices, and all Mac computers with the Apple T2 Security Chip. The Secure Enclave provides the foundation for encrypting data at rest, secure boot in macOS, and biometrics.

All modern iPhone, iPad, and Mac computers with a T2 chip include a dedicated AES hardware engine to power line-speed encryption as files are written or read. This ensures that Data Protection and FileVault protect users' files without exposing long-lived encryption keys to the CPU or operating system. For more information on which Apple hardware contains the Secure Enclave, see the Secure Enclave overview.

Secure boot of Apple devices ensures that the lowest levels of software aren't tampered with and that only trusted operating system software from Apple loads at startup. In iOS and iPadOS devices, security begins in immutable code called the Boot ROM, which is laid down during chip fabrication and known as the *hardware root of trust*. On Mac computers with a T2 chip, trust for macOS secure boot begins with the T2 chip itself. (Both the T2 and Secure Enclave also execute their own secure boot processes.)

The Secure Enclave enables Touch ID and Face ID in Apple devices to provide secure authentication while keeping user biometric data private and secure. This enables users to enjoy the security of longer and more complex passcodes and passwords with, in many situations, the convenience of swift authentication.

The security features of Apple devices are made possible by the combination of silicon design, hardware, software, and services available only from Apple.

Secure Enclave

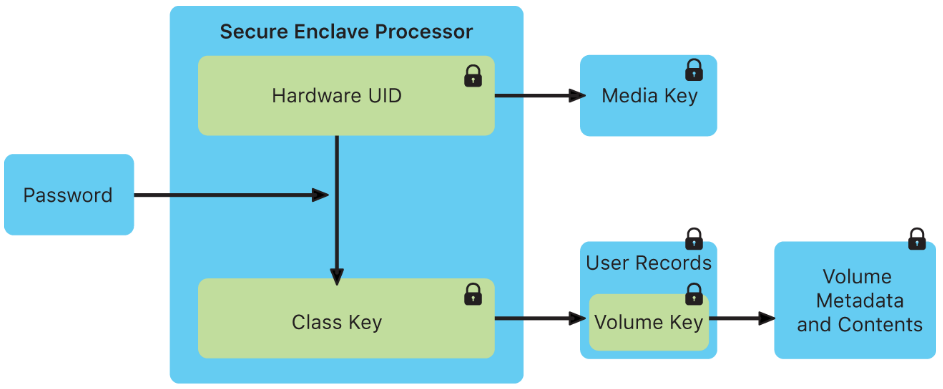
Secure Enclave overview

The Secure Enclave is a hardware-based key manager that's isolated from the main processor to provide an extra layer of security. The Secure Enclave is a hardware feature of certain versions of iPhone, iPad, Mac, Apple TV, Apple Watch, and HomePod—namely:

- iPhone 5s (or later)
- iPad Air (or later)
- Mac computers that contain the Apple T2 Security Chip
- Apple TV 4th generation (or later)
- Apple Watch Series 1 (or later)
- HomePod

The key data is encrypted in the Secure Enclave system on chip (SoC), which includes a random number generator.

The Secure Enclave also maintains the integrity of its cryptographic operations even if the device kernel has been compromised. Communication between the Secure Enclave and the application processor is tightly controlled by isolating it to an interrupt-driven mailbox and shared memory data buffers.



The Secure Enclave processor.

Dedicated Boot ROM

The Secure Enclave includes a dedicated Secure Enclave Boot ROM. Similar to the application processor Boot ROM, the Secure Enclave Boot ROM is immutable code that establishes the hardware root of trust for the Secure Enclave. It also runs a Secure Enclave OS based on an Apple-customized version of the L4 microkernel. This Secure Enclave OS is signed by Apple, verified by the Secure Enclave Boot ROM, and updated through a personalized software update process.

When the device starts up, an ephemeral memory protection key is created by the Secure Enclave Boot ROM, entangled with the device's unique ID (UID), and used to encrypt the Secure Enclave's portion of the device's memory space. Except on the Apple A7, the Secure Enclave memory is also authenticated with the memory protection key. On A11 (and newer) and S4 SoCs, an integrity tree is used to prevent replay of security-critical Secure Enclave memory, authenticated by the memory protection key and nonces stored in on-chip SRAM.

In iOS and iPadOS, when the Secure Enclave saves files to the file system, all files are encrypted with a key entangled with the UID and an anti-replay nonce. On A9 (and newer) SoCs, the anti-replay nonce uses entropy generated by the hardware random number generator. The anti-replay nonce support is rooted in a dedicated nonvolatile memory integrated circuit (IC). In Mac computers, the FileVault key hierarchy is similarly linked to the UID of the Secure Enclave.

In devices with A12 (and newer) and S4 SoCs, the Secure Enclave is paired with a secure storage IC for anti-replay nonce storage. The secure storage IC is designed with immutable ROM code, a hardware random number generator, cryptography engines, and physical tamper detection. To read and update nonces, the Secure Enclave and storage IC employ a secure protocol that ensures exclusive access to the nonces.

Anti-replay services

Anti-replay services on the Secure Enclave are used for revocation of data over events that mark anti-replay boundaries including, but not limited to, the following:

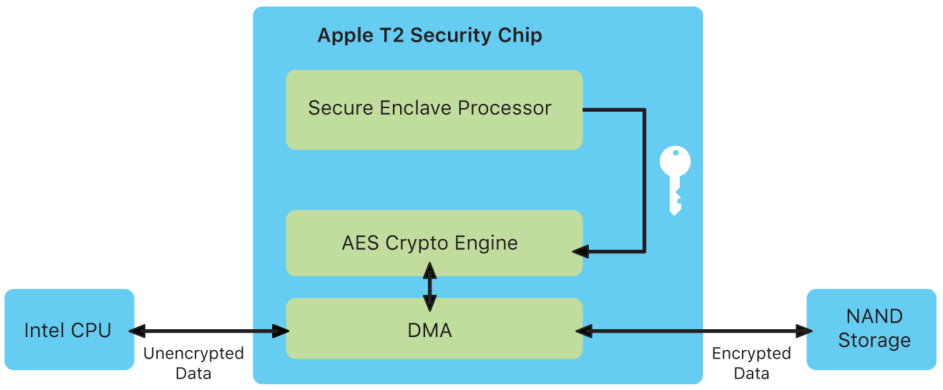
- Passcode change
- Enabling or disabling Touch ID or Face ID
- Adding or removing a Touch ID fingerprint or Face ID face
- Touch ID or Face ID reset
- Adding or removing an Apple Pay card
- Erase All Content and Settings

Dedicated AES engine

Every Apple device with a Secure Enclave has a dedicated AES-256 crypto engine built into the DMA path between the flash storage and main system memory, making file encryption highly efficient. On A9 or later A-series processors, the flash storage subsystem is on an isolated bus that is only granted access to memory containing user data through the DMA crypto engine.

The Secure Enclave securely generates its own unique keys—unique IDs (UIDs), device group IDs (GIDs), and other—and securely erases saved keys when needed. These keys are AES-256-bit keys fused (the UID) or compiled (the GID) into the Secure Enclave during manufacturing. No software or firmware can read them directly; they can see only the results of encryption or decryption operations performed by dedicated AES engines implemented in silicon using those UIDs or GIDs as a key.

The application processor and Secure Enclave each have their own UID and GID, and the Secure Enclave UID and GID can be used only by the AES engine dedicated to the Secure Enclave. The UIDs and GIDs aren't available through Joint Test Action Group (JTAG) or other debugging interfaces.



The AES Cryptographic engine supports line-speed encryption on the DMA path on Mac computers with the Apple T2 Security Chip.

Generating cryptographic keys

Each Secure Enclave generates its own UID (Unique ID) during the manufacturing process. Because the UID is unique to each device and because it's generated wholly within the Secure Enclave instead of in a manufacturing system outside of the device, the UID isn't available for access or storage by Apple or any of its suppliers. This applies to all SoCs after the Apple A8 processor.

Software running on the Secure Enclave takes advantage of the UID to protect device-specific secrets. The UID allows data to be cryptographically tied to a particular device. For example, the key hierarchy protecting the file system includes the UID, so if the internal SSD storage is physically moved from one device to another, the files are inaccessible. The UID isn't related to any other identifier on the device. Other protected device-specific secrets include Touch ID or Face ID data. Storage on devices not connected to the Apple T2 Security Chip don't receive this level of encryption. For example, neither external storage devices connected over USB nor PCIe-based storage added to the 2019 Mac Pro are encrypted by the T2 chip.

At the device level is the device group ID (GID), which is common to all processors in a class of devices (for example, all devices using the Apple A8 processor).

Apart from the UID and GID, all other cryptographic keys in iOS and iPadOS devices are created by the system's random number generator (RNG) using an algorithm based on CTR_DRBG. System entropy is generated from timing variations during boot, and additionally from interrupt timing after the device has booted. Keys generated inside the Secure Enclave use its true hardware random number generator based on multiple ring oscillators post processed with CTR_DRBG.

Secure data erasure

Securely erasing saved keys is just as important as generating them. It's especially challenging to do so on flash storage, for example, where wear-leveling might mean that multiple copies of data need to be erased. To address this issue, devices with a Secure Enclave include a feature dedicated to secure data erasure called Effaceable Storage. This feature accesses the underlying storage technology (for example, NAND) to directly address and erase a small number of blocks at a very low level.

Touch ID and Face ID

Touch ID and Face ID overview

Passcodes and password are critical to the security of Apple devices, but users need to be able to quickly access their devices—even upwards of a hundred times a day. Biometric authentication provides an opportunity to retain the security of a strong passcode—or even to strengthen the passcode or password since it won't often need to be entered manually—while providing the convenience of swiftly unlocking with a finger press or glance. Touch ID and Face ID don't replace the password or passcode, but they do make access faster and easier in most situations.

Touch ID security

Touch ID is the fingerprint sensing system that makes secure access to supported Apple devices faster and easier. This technology reads fingerprint data from any angle and learns more about a user's fingerprint over time, with the sensor continuing to expand the fingerprint map as additional overlapping nodes are identified with each use.

Apple devices with a Touch ID sensor can be unlocked using a fingerprint. Touch ID doesn't replace the need for a device passcode or user password, which is still required after device startup, restart, or logout (on a Mac). In some apps, Touch ID can also be used in place of device passcode or user password—for example, to unlock password protected notes in the Notes app, to unlock keychain-protected websites, and to unlock supported app passwords. However, a device passcode or user password is always required in some scenarios. For example to change an existing device passcode or user password or to remove existing fingerprint enrollments or create new ones.

When the fingerprint sensor detects the touch of a finger, it triggers the advanced imaging array to scan the finger and sends the scan to the Secure Enclave. Communication between the processor and the Touch ID sensor takes place over a serial peripheral interface bus. The processor forwards the data to the Secure Enclave but can’t read it. It’s encrypted and authenticated with a session key that is negotiated using a shared key provisioned for each Touch ID sensor and its corresponding Secure Enclave at the factory. The shared key is strong, random, and different for every Touch ID sensor. The session key exchange uses AES key wrapping, with both sides providing a random key that establishes the session key and uses AES-CCM transport encryption.

While being vectorized for analysis, the raster scan is temporarily stored in encrypted memory within the Secure Enclave and then it is discarded. The analysis utilizes subdermal ridge flow angle mapping, which is a lossy process that discards minutiae data that would be required to reconstruct the user’s actual fingerprint. The resulting map of nodes is stored without any identity information in an encrypted format that can only be read by the Secure Enclave. This data never leaves the device. It’s not sent to Apple, nor is it included in device backups.

Face ID security

With a simple glance, Face ID securely unlocks supported Apple devices. It provides intuitive and secure authentication enabled by the TrueDepth camera system, which uses advanced technologies to accurately map the geometry of a user’s face. Face ID uses neural networks for determining attention, matching, and anti-spoofing, so a user can unlock their phone with a glance. Face ID automatically adapts to changes in appearance, and carefully safeguards the privacy and security of a user’s biometric data.

Face ID is designed to confirm user attention, provide robust authentication with a low false-match rate, and mitigate both digital and physical spoofing.

The TrueDepth camera automatically looks for the user’s face when they wake Apple devices that feature Face ID (by raising it or tapping the screen), as well as when those devices attempt to authenticate the user in order to display an incoming notification or when a supported app requests Face ID authentication. When a face is detected, Face ID confirms attention and intent to unlock by detecting that the user’s eyes are open and their attention is directed at their device; for accessibility, this is disabled when VoiceOver is activated and, if required, can be disabled separately.

After it confirms the presence of an attentive face, the TrueDepth camera projects and reads over 30,000 infrared dots to form a depth map of the face, along with a 2D infrared image. This data is used to create a sequence of 2D images and depth maps, which are digitally signed and sent to the Secure Enclave. To counter both digital and physical spoofs, the TrueDepth camera randomizes the sequence of 2D images and depth map captures, and projects a device-specific random pattern. A portion of the SoCs neural engine—protected within the Secure Enclave—transforms this data into a mathematical representation and compares that representation to the enrolled facial data. This enrolled facial data is itself a mathematical representation of the user’s face captured across a variety of poses.

Touch ID, Face ID, passcodes, and passwords

To use Touch ID or Face ID, the user must set up their device so that a passcode or password is required to unlock it. When Touch ID or Face ID detects a successful match, the user's device unlocks without asking for the device passcode or password. This makes using a longer, more complex passcode or password far more practical because the user doesn't need to enter it as frequently. Touch ID and Face ID don't replace the user's passcode or password, but provide easy access to the device within thoughtful boundaries and time constraints. This is important because a strong passcode or password forms the foundation for how the user's iOS, iPadOS, macOS, or watchOS device cryptographically protects their data.

When a device passcode or password is required

Users can use their passcode or password anytime instead of Touch ID or Face ID, but there are some situations where biometrics aren't permitted. The following security-sensitive operations always require entry of a passcode or password:

- Updating the software
- Erasing the device
- Viewing or changing passcode settings
- Installing configuration profiles
- Unlocking the Security & Privacy preferences pane in System Preferences on Mac
- Unlocking the Users & Groups preferences pane in System Preferences on Mac (if FileVault is turned on)

A passcode or password is also required if the device is in the following states:

- The device has just been turned on or restarted
- The user has logged out of their Mac account (or has not yet logged in)
- The user has not unlocked their device for more than 48 hours
- The user hasn't used their passcode or password to unlock their device for 156 hours (six and a half days) and the user hasn't used a biometric to unlock their device in 4 hours
- The device has received a remote lock command
- After exiting power off/Emergency SOS by pressing and holding either volume button and the sleep/wake simultaneously for 2 seconds and then pressing Cancel
- After five unsuccessful biometric match attempts (though for usability, the device might offer entering a passcode or password instead of using biometrics after a smaller number of failures)

When Touch ID or Face ID is enabled on an iPhone or iPad, the device immediately locks when the sleep/wake is pressed, and the device locks every time it goes to sleep. Touch ID and Face ID require a successful match—or optionally the passcode—at every wake.

The probability that a random person in the population could unlock a user's iPhone, iPad, or Mac is 1 in 50,000 with Touch ID or 1 in 1,000,000 with Face ID. This probability increases with multiple enrolled fingerprints (up to 1 in 10,000 with five fingerprints) or appearances (up to 1 in 500,000 with two appearances). For additional protection, both Touch ID and Face ID allow only five unsuccessful match attempts before a passcode or password is required to obtain access to the user's device or account. With Face ID, the probability of a false match is different for twins and siblings who look like the user and for children under the age of 13 (because their distinct facial features may not have fully developed). If the user is concerned about this, Apple recommends using a passcode to authenticate.

Facial matching

Facial matching is performed within the Secure Enclave using neural networks trained specifically for that purpose. Apple developed the facial matching neural networks using over a billion images, including IR and depth images collected in studies conducted with the participants' informed consent. Apple then worked with participants from around the world to include a representative group of people accounting for gender, age, ethnicity, and other factors. The studies were augmented as needed to provide a high degree of accuracy for a diverse range of users. Face ID is designed to work with hats, scarves, eyeglasses, contact lenses, and many sunglasses. Furthermore, it's designed to work indoors, outdoors, and even in total darkness. An additional neural network that's trained to spot and resist spoofing defends against attempts to unlock the device with photos or masks. Face ID data, including mathematical representations of a user's face, is encrypted and available only to the Secure Enclave. This data never leaves the device. It's not sent to Apple, nor is it included in device backups. The following Face ID data is saved, encrypted only for use by the Secure Enclave, during normal operation:

- The mathematical representations of a user's face calculated during enrollment
- The mathematical representations of a user's face calculated during some unlock attempts if Face ID deems them useful to augment future matching

Face images captured during normal operation aren't saved, but are instead immediately discarded after the mathematical representation is calculated for either enrollment or comparison to the enrolled Face ID data.

Improving Face ID matches

To improve match performance and keep pace with the natural changes of a face and look, Face ID augments its stored mathematical representation over time. Upon successful match, Face ID may use the newly calculated mathematical representation—if its quality is sufficient—for a finite number of additional matches before that data is discarded. Conversely, if Face ID fails to recognize a face, but the match quality is higher than a certain threshold and a user immediately follows the failure by entering their passcode, Face ID takes another capture and augments its enrolled Face ID data with the newly calculated mathematical representation. This new Face ID data is discarded if the user stops matching against it and after a finite number of matches. These augmentation processes allow Face ID to keep up with dramatic changes in a user's facial hair or makeup use, while minimizing false acceptance.

Face ID Diagnostics

Face ID data doesn’t leave the device and is never backed up to iCloud or anywhere else. Only when the user wants to provide Face ID diagnostic data to AppleCare for support is this information transferred from the user’s device. Enabling Face ID Diagnostics requires a digitally signed authorization from Apple that is similar to the one used in the software update personalization process. After authorization, the user can activate Face ID Diagnostics and begin the setup process from within the Settings app on devices that support Face ID.

As part of setting up Face ID Diagnostics, the existing Face ID enrollment is deleted and the user is asked to reenroll in Face ID. Devices that support Face ID begin recording Face ID images captured during authentication attempts for the next 10 days and automatically stop saving images thereafter. Face ID Diagnostics doesn’t automatically send data to Apple. The user can review and approve enrollment and unlock images (both successful and failed) included in Face ID diagnostic data that is gathered while in diagnostics mode before it’s sent to Apple. Face ID Diagnostics uploads only the Face ID Diagnostics images the user approves. The data is encrypted before it’s uploaded and is immediately deleted from the device after the upload is complete. Images the user rejects are immediately deleted.

If the user doesn’t conclude the Face ID Diagnostics session by reviewing images and uploading any approved images, Face ID Diagnostics automatically ends after 40 days and all diagnostic images are deleted from the device. Users can also disable Face ID Diagnostics at any time. All local images are immediately deleted if the user does so, and no Face ID data is shared with Apple in these cases.

Unlocking a device or user account

With Touch ID or Face ID disabled, when a device or account locks, the keys for the highest class of Data Protection—which are held in the Secure Enclave—are discarded. The files and Keychain items in that class are inaccessible until the user unlocks the device or account by entering their passcode or password.

With Touch ID or Face ID enabled, the keys aren’t discarded when the device or account locks; instead, they’re wrapped with a key that’s given to the Touch ID or Face ID subsystem inside the Secure Enclave. When a user attempts to unlock the device or account, if the device detects a successful match, it provides the key for unwrapping the Data Protection keys, and the device or account is unlocked. This process provides additional protection by requiring cooperation between the Data Protection and Touch ID or Face ID subsystems to unlock the device.

When the device restarts, the keys required for Touch ID or Face ID to unlock the device or account are lost; they’re discarded by the Secure Enclave after any condition is met that requires passcode or password entry.

Securing purchases with Apple Pay

The user can also use Touch ID and Face ID with Apple Pay to make easy and secure purchases in stores, apps, and on the web.

To authorize an in-store payment with Face ID, the user must first confirm intent to pay by double-clicking the side button. This double-click captures user intent using a physical gesture directly linked to the Secure Enclave and is resistant to forgery by a malicious process. The user then authenticates using Face ID before placing the device near the contactless payment reader. A different Apple Pay payment method can be selected after Face ID authentication which requires reauthentication, but the user won't have to double-click the side button again.

To make a payment within apps and on the web, the user confirms their intent to pay by double-clicking the side button, then authenticates using Face ID to authorize the payment. If the Apple Pay transaction isn't completed within 60 seconds of double-clicking the side button, the user must reconfirm intent to pay by double-clicking again.

In the case of Touch ID, the intent to pay is confirmed using the gesture of activating the Touch ID sensor combined with successfully matching the user's fingerprint.

Other uses for Touch ID and Face ID

Third-party apps can use system-provided APIs to ask the user to authenticate using Touch ID or Face ID or a passcode or password, and apps that support Touch ID automatically support Face ID without any changes. When using Touch ID or Face ID, the app is notified only as to whether the authentication was successful; it can't access Touch ID, Face ID, or the data associated with the enrolled user.

Protecting Keychain items

Keychain items can also be protected with Touch ID or Face ID, to be released by the Secure Enclave only by a successful match or the device passcode or account password. App developers have APIs to verify that a passcode or password has been set by the user, before requiring Touch ID or Face ID or a passcode or password to unlock Keychain items. App developers can do the following:

- Require that authentication API operations does not fall back to an app password or the device passcode. They can query whether a user is enrolled, allowing Touch ID or Face ID to be used as a second factor in security-sensitive apps.
- Generate and use ECC keys inside Secure Enclave that can be protected by Touch ID or Face ID. Operations with these keys are always performed inside the Secure Enclave after it authorizes their use.

Making and approving purchases

Users can also configure Touch ID or Face ID to approve purchases from the iTunes Store, the App Store, Apple Books, and more, so users don't have to enter their Apple ID password. With iOS 11 or later or macOS 10.12.5 or later, Touch ID– and Face ID–protected Secure Enclave ECC keys are used to authorize a purchase by signing the store request.

Hardware microphone disconnect in Mac

All Mac portables with the Apple T2 Security Chip feature a hardware disconnect that ensures the microphone is disabled whenever the lid is closed. On the 13-inch MacBook Pro and MacBook Air computers with the T2 chip, and on the 15-inch MacBook Pro portables from 2019 or later, this disconnect is implemented in hardware alone. The disconnect prevents any software—even with root or kernel privileges in macOS, and even the software on the T2 chip—from engaging the microphone when the lid is closed. (The camera is not disconnected in hardware, because its field of view is completely obstructed with the lid closed.)

Express Cards with power reserve in iPhone

If iOS isn’t running because iPhone needs to be charged, there may still be enough power in the battery to support Express Card transactions. Supported iPhone devices automatically support this feature with:

- A transit card designated as the Express Transit card
- Student ID cards with Express Mode turned on

Pressing the side button displays the low battery icon as well as text indicating Express Cards are available to use. The NFC controller performs Express Card transactions under the same conditions as when iOS is running, except that transactions are indicated with only haptic notification. No visible notification is shown.

This feature isn’t available when a standard user initiated shutdown is performed.

System Security

System security overview

Building on the unique capabilities of Apple hardware, system security is designed to maximize the security of the operating systems on Apple devices without compromising usability. System security encompasses the boot-up process, software updates, and the ongoing operation of the OS.

Secure boot begins in hardware and builds a chain of trust through software, where each step ensures that the next is functioning properly before handing over control. This security model supports not only the default boot of Apple devices but also the various modes for recovery and updating on iOS, iPadOS, and macOS devices.

The most recent versions of iOS, iPadOS, or macOS are the most secure. The software update mechanism not only provides timely updates to Apple devices—it also is delivers only known good software from Apple. The update system can even prevent downgrade attacks, so devices can't be rolled back to an older version of the operating system (which an attacker knows how to compromise) as a method of stealing user data.

Finally, Apple devices include boot and runtime protections so that they maintain their integrity during ongoing operation. These protections vary significantly between iOS, iPadOS, and macOS devices based on the very different sets of capabilities they support and the attacks they must therefore thwart.

Random number generation

Cryptographic pseudorandom number generators (CPRNGs) are an important building block for secure software. To this end, Apple provides a trusted software CPRNG running in the iOS, iPadOS, macOS, tvOS, and watchOS kernels. It's responsible for aggregating raw entropy from the system and providing secure random numbers to consumers in both the kernel and user space.

Entropy sources

The kernel CPRNG is seeded from multiple entropy sources during boot and over the lifetime of the device. These include (contingent on availability):

- The Secure Enclave's hardware RNG
- Timing-based jitter collected during boot
- Entropy collected from hardware interrupts

- A seed file used to persist entropy across boots
- Intel random instructions, i.e. RDSEED and RDRAND (macOS-only)

The Kernel CPRNG

The kernel CPRNG is a Fortuna-derived design targeting a 256-bit security level. It provides high-quality random numbers to user-space consumers using the following APIs:

- The `getentropy` (2) system call
- The random device, i.e. /dev/random

The kernel CPRNG accepts user-supplied entropy through writes to the random device.

Secure boot

iOS and iPadOS secure boot chain

Each step of the startup process contains components that are cryptographically signed by Apple to ensure integrity and that proceed only after verifying the chain of trust. This includes the bootloaders, the kernel, kernel extensions, and baseband firmware. This secure boot chain helps ensure that the lowest levels of software aren't tampered with.

When an iOS or iPadOS device is turned on, its application processor immediately executes code from read-only memory referred to as Boot ROM. This immutable code, known as the hardware root of trust, is laid down during chip fabrication and is implicitly trusted. The Boot ROM code contains the Apple Root CA public key—used to verify that the iBoot bootloader is signed by Apple before allowing it to load. This is the first step in the chain of trust, in which each step ensures that the next is signed by Apple. When the iBoot finishes its tasks, it verifies and runs the iOS or iPadOS kernel. For devices with an A9 or earlier A-series processor, an additional Low-Level Bootloader (LLB) stage is loaded and verified by the Boot ROM and in turn loads and verifies iBoot.

A failure to load or verify following stages is handled differently depending on the hardware:

- *Boot ROM can't load LLB (older devices):* Device Firmware Upgrade (DFU) mode
- *LLB or iBoot:* Recovery mode

In either case, the device must be connected to iTunes (in macOS 10.14 or earlier) or the Finder (macOS 10.15 or later) through USB and restored to factory default settings.

The Boot Progress Register (BPR) is used by the Secure Enclave to limit access to user data in different modes and is updated before entering the following modes:

- *DFU mode:* Set by Boot ROM on devices with an Apple A12 or newer SoCs
- *Recovery mode:* Set by iBoot on devices with Apple A10, S2, or newer SoCs

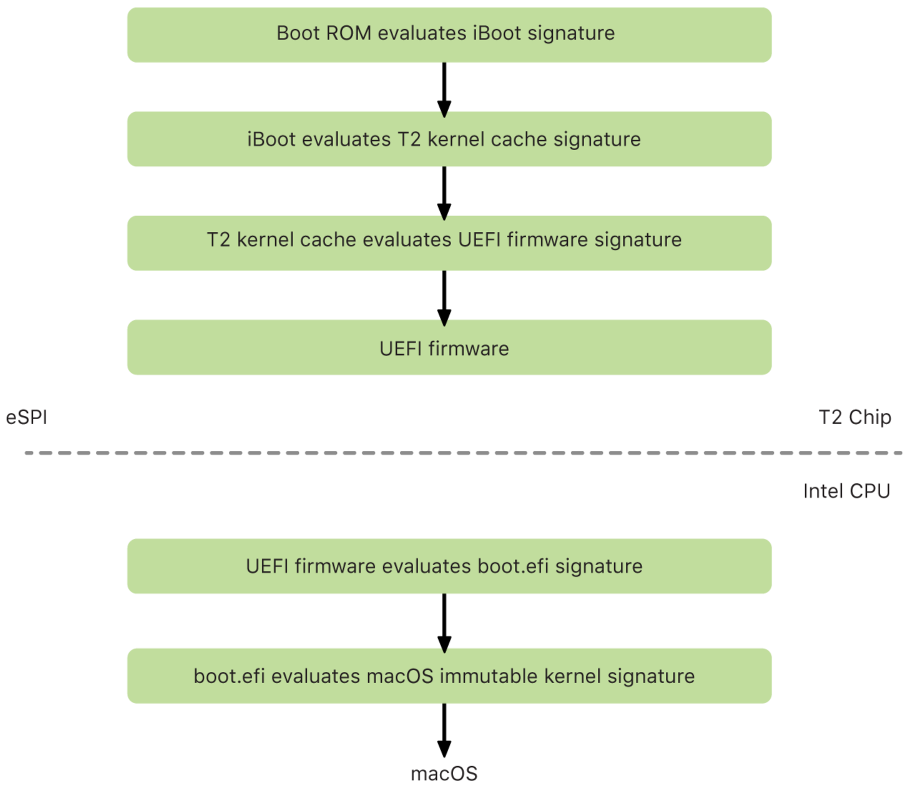
On devices with cellular access, the baseband subsystem also utilizes its own similar process of secure booting with signed software and keys verified by the baseband processor.

The Secure Enclave coprocessor also utilizes a secure boot process that ensures its separate software is verified and signed by Apple.

macOS boot modes

Boot process of Mac computers

When a Mac computer with the Apple T2 Security Chip chip is turned on, the chip executes code from read-only memory known as Boot ROM. This immutable code, referred to as the *hardware root of trust*, is laid down during chip fabrication and is audited for vulnerabilities and implicitly trusted. The Boot ROM code contains the Apple Root CA public key, which is used to verify that the iBoot bootloader is signed by Apple's private key before allowing it to load. This is the first step in the chain of trust. iBoot verifies the kernel and kernel extension code on the T2 chip, which subsequently verifies the Intel UEFI firmware. The UEFI firmware and the associated signature are initially available only to the T2 chip.


Boot ROM evaluates iBoot signature
iBoot evaluates T2 kernel cache signature
T2 kernel cache evaluates UEFI firmware signature
UEFI firmware
eSPI
T2 Chip
Intel CPU
UEFI firmware evaluates boot.efi signature
boot.efi evaluates macOS immutable kernel signature
macOS

macOS secure boot chain.

After verification, the UEFI firmware image is mapped into a portion of the T2 chip memory. This memory is made available to the Intel CPU through the enhanced Serial Peripheral Interface (eSPI). When the Intel CPU first boots, it fetches the UEFI firmware through eSPI from the integrity-checked, memory-mapped copy of the firmware located on the T2 chip.

The evaluation of the chain of trust continues on the Intel CPU, with the UEFI firmware evaluating the signature for boot.efi, which is the macOS bootloader. The Intel-resident macOS secure boot signatures are stored in the same Image4 format used for iOS, iPadOS, and T2 chip secure boot, and the code that parses the Image4 files is the same hardened code from the current iOS and iPadOS secure boot implementation. Boot.efi in turn verifies the signature of a new file, called immutablekernel. When secure boot is enabled, the immutablekernel file represents the complete set of Apple kernel extensions required to boot macOS. The secure boot policy terminates at the handoff to the immutablekernel, and after that, macOS security policies (such as System Integrity Protection and signed kernel extensions) take effect.

If there are any errors or failures in this process, the Mac enters macOS Recovery mode, Apple T2 Security Chip Recovery mode, or Apple T2 Security Chip DFU mode.

Boot modes overview of Mac computers

Mac computers have a variety of boot modes that can be entered at boot time by pressing key combinations, which are recognized by the UEFI firmware or booter. Some boot modes, such as Single User Mode, won’t work unless the security policy is changed to No Security in Startup Security Utility.

Mode	Key combo	Description
macOS boot	None	The UEFI firmware hands off to the macOS booter (a UEFI application) which hands off to the macOS kernel. On standard booting of a Mac with FileVault enabled, the macOS booter is the code presenting the Login Window interface in order to take the password to decrypt the storage.
Startup Manager	Option (⌥)	The UEFI firmware launches the built-in UEFI application which presents the user with a boot device selection interface.
Target Disk Mode (TDM)	T	The UEFI firmware launches the built-in UEFI application which exposes the internal storage device as a raw, block-based storage device over FireWire, Thunderbolt, USB, or any combination of the three (depending on the model of the Mac).

Mode	Key combo	Description
Single User Mode	Command (⌘)-S	The macOS kernel passes the `-s` flag in launchd's argument vector, then launchd creates the single-user shell in the Console app's tty. *Note:* If the user exits the shell, macOS continues boot to the Login window.
RecoveryOS	Command (⌘)-R	The UEFI firmware loads a minimal macOS from a signed disk image (.dmg) file on the internal storage device.
Internet RecoveryOS	Option (⌥)-Command (⌘)-R	The signed disk image is downloaded from the internet using HTTP.
Diagnostics	D	The UEFI firmware loads a minimal UEFI diagnostic environment from a signed disk image file on the internal storage device.
Internet Diagnostics	Option (⌥)-D	The signed disk image is downloaded from the internet using HTTP.
Netboot (For Mac computers without an Apple T2 Security Chip)	N	The UEFI firmware downloads the macOS booter from a local TFTP server, the booter downloads the macOS kernel from the same TFTP server, and the macOS kernel mounts a filesystem from an NFS or HTTP network share.
Windows boot	None	If Windows has been installed using BootCamp, the UEFI firmware hands off to the Windows booter, which hands off to the Windows kernel.

recoveryOS and diagnostics environments in Mac computers

The recoveryOS is completely separate from the main macOS, and the entire contents are stored in a disk image file named BaseSystem.dmg. There is also an associated BaseSystem.chunklist which is used to verify the integrity of the BaseSystem.dmg. The chunklist is a series of hashes for 10 MB chunks of the BaseSystem.dmg. The UEFI firmware evaluates the signature of the chunklist file, and then evaluates the hash for one chunk at a time from the BaseSystem.dmg, to ensure that it matches the signed content present in the chunklist. If any of these hashes does not match, booting from the local recovery OS is aborted, and the UEFI firmware attempts to boot from Internet Recovery instead.

If the verification completes successfully, the UEFI firmware mounts the BaseSystem.dmg as a ramdisk and launches the boot.efi contained therein. There is no need for the UEFI firmware to do a specific check of the boot.efi, nor for the boot.efi to do a check of the kernel, because the completed contents of the OS (of which these elements are only a subset) have already been integrity checked.

The procedure for booting the local diagnostic environment is mostly the same as launching the recoveryOS. Separate AppleDiagnostics.dmg and AppleDiagnostics.chunklist are used, but they are verified the same way as the BaseSystem files. Instead of launching boot.efi, the UEFI firmware launches a file inside the dmg named diags.efi, which is in turn responsible for invoking a variety of other UEFI drivers that can interface with and check for errors in the hardware.

Internet recoveryOS and diagnostics environments in Mac computers

If an error has occurred in the launching of the local recovery or diagnostic environments, the UEFI firmware attempts to download the images from the Internet instead. Additionally, a user can request that the images be fetched from the Internet using special key sequences held at boot. The integrity validation of the disk images and chunklists downloaded from the OS Recovery Server is performed the same way as with images retrieved from a storage device.

While the connection to the OS Recovery Server is done using HTTP, the complete downloaded contents are still integrity checked as previously described, and as such are not vulnerable to manipulation by an attacker with control of the network. In the event that an individual chunk fails integrity verification, it's requested again from the OS Recovery Server 11 times, before giving up and displaying an error.

Microsoft Windows boot in Mac computers

By default, Mac computers that support secure boot trust only content signed by Apple. However, to improve the security of Boot Camp installations, Apple also supports secure booting for Windows. The UEFI firmware includes a copy of the Microsoft Windows Production CA 2011 certificate used to authenticate Microsoft bootloaders.

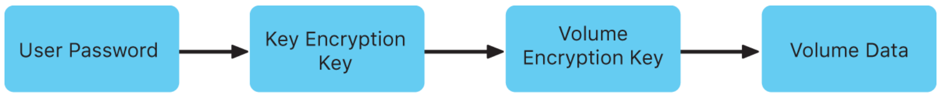
Note: There is currently no trust provided for the Microsoft Corporation UEFI CA 2011, which would allow verification of code signed by Microsoft partners. This UEFI CA is commonly used to verify the authenticity of bootloaders for other operating systems, such as Linux variants.

Support for secure boot of Windows isn't enabled by default; instead, it's enabled using Boot Camp Assistant (BCA). When a user runs BCA, macOS is reconfigured to trust Microsoft first-party signed code during boot. After BCA completes, if macOS fails to pass the Apple first-party trust evaluation during secure boot, the UEFI firmware attempts to evaluate the trust of the object according to UEFI Secure Boot formatting. If the trust evaluation succeeds, the Mac proceeds and boots Windows. If not, the Mac enters macOS Recovery and informs the user of the trust evaluation failure.

Boot process of Mac computers without an Apple T2 Security Chip

Mac computers without an Apple T2 Security Chip don't support secure boot. Therefore the UEFI firmware loads the macOS booter (boot.efi) from the filesystem without verification, and the booter loads the kernel (prelinkedkernel) from the filesystem without verification. To protect the integrity of the boot chain, users should enable all of the following security mechanisms:

- *System Integrity Protection:* Enabled by default, this protects the booter and kernel against malicious writes from within a running macOS.
- *FileVault:* This can be enabled in two ways: by the user or by a mobile device management (MDM) administrator. This protects against a physically present attacker using Target Disk Mode to overwrite the booter.
- *Firmware Password:* This can be enabled in two ways: by the user or by a mobile device management (MDM) administrator. This protects a physically present attacker from launching alternate boot modes such as recoveryOS, Single User Mode, or Target Disk Mode from which the booter can be overwritten. This also prevents booting from alternate media, by which an attacker could run code to overwrite the booter.

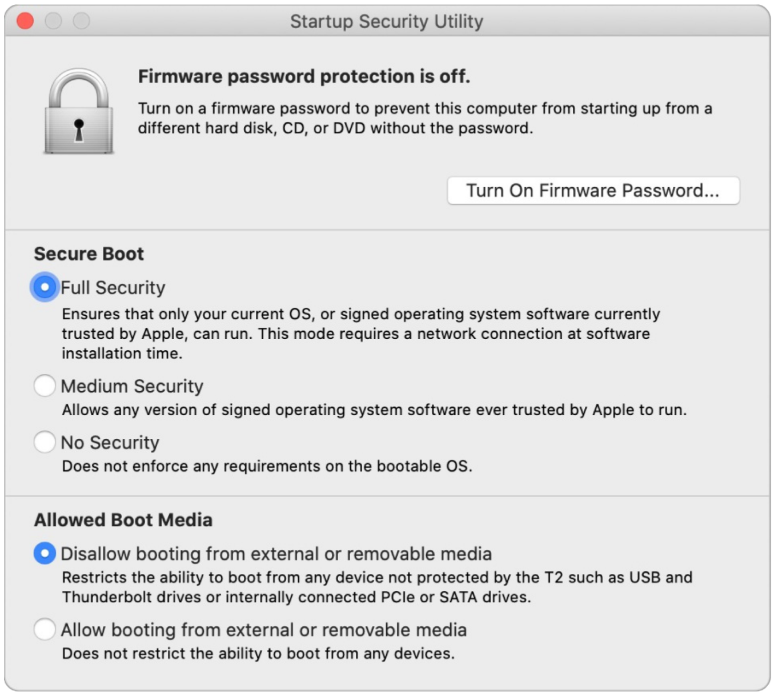


Unlocking process of Mac computers without an Apple T2 Security Chip.

Startup Security Utility

Startup Security Utility overview

Startup Security Utility is a replacement to the previous Firmware Password Utility. On Mac computers with an Apple T2 Security Chip, it handles a larger set of security policy settings. Mac computers without a T2 chip continue to use Firmware Password Utility. The utility is accessible by booting into recoveryOS and selecting Startup Security Utility from the Utilities menu. The advantage of putting critical system security policy controls (such as secure boot or SIP) in the recoveryOS is that the entire OS is integrity checked. This ensures that any attacker code that has broken into the Mac can't trivially impersonate the user for purposes of further disabling security policies.


Startup Security Utility

Firmware password protection is off.

Turn on a firmware password to prevent this computer from starting up from a different hard disk, CD, or DVD without the password.

Secure Boot

[✓ Full Security]
Ensures that only your current OS, or signed operating system software currently trusted by Apple, can run. This mode requires a network connection at software installation time.

Medium Security
Allows any version of signed operating system software ever trusted by Apple to run.

No Security
Does not enforce any requirements on the bootable OS.

Allowed Boot Media

[✓ Disallow booting from external or removable media]
Restricts the ability to boot from any device not protected by the T2 such as USB and Thunderbolt drives or internally connected PCIe or SATA drives.

Allow booting from external or removable media
Does not restrict the ability to boot from any devices.

Startup Security Utility.

Critical policy changes now require authentication, even in Recovery mode. This feature is available only on Mac computers containing the T2 chip. When Startup Security Utility is first opened, it prompts the user to enter an administrator password from the primary macOS installation associated with the currently booted macOS Recovery. If no administrator exists, one must be created before the policy can be changed. The T2 chip requires that the Mac computer is currently booted into macOS Recovery and that an authentication with a Secure Enclave–backed credential has occurred before such a policy change can be made. Security policy changes have two implicit requirements. macOS Recovery must:

- Be booted from a storage device directly connected to the T2 chip, because partitions on other devices don’t have Secure Enclave–backed credentials bound to the internal storage device.
- Reside on an APFS-based volume, because there is support only for storing the Authentication in Recovery credentials sent to the Secure Enclave on the “Preboot” APFS volume of a drive. HFS plus-formatted volumes can’t use secure boot.

This policy is only shown in Startup Security Utility on Mac computers with an Apple T2 Security Chip. Although the majority of use cases shouldn’t require changes to the secure boot policy, users are ultimately in control of their device’s settings, and may choose, depending on their needs, to disable or downgrade the secure boot functionality on their Mac.

Secure boot policy changes made from within this app apply only to the evaluation of the chain of trust being verified on the Intel processor. The option “Secure boot the T2 chip” is always in effect.

Secure boot policy can be configured to one of three settings: Full Security, Medium Security, and No Security. No Security completely disables secure boot evaluation on the Intel processor and allows the user to boot whatever they want.

Full Security boot policy

Full Security is the default, and it behaves like iOS and iPadOS. At the time that software is downloaded and prepared to install, rather than using the global signature which comes with the software, macOS talks to the same Apple signing server used for iOS and iPadOS and requests a fresh, “personalized” signature. A signature is said to be personalized when it includes the ECID—a unique ID specific to the T2 chip in this case—as part of the signing request. The signature which is given back by the signing server is then unique and usable only by that particular T2 chip. When the Full Security policy is in effect, the UEFI firmware ensures that a given signature isn’t just signed by Apple, but is signed for this specific Mac, essentially tying that version of macOS to that Mac.

Using an online signing server also provides better protection against rollback attacks than typical global signature approaches. In a global signing system, the security epoch could have rolled many times, but a system that has never seen the latest firmware won’t know this. For example, a computer that currently believes it is in security epoch 1 accepts software from security epoch 2, even if the current actual security epoch is 5. With an iOS and iPadOS type of online signing system, the signing server can reject creating signatures for software which is in anything except the latest security epoch.

Additionally, if an attacker discovers a vulnerability after a security epoch change, they can’t simply pick up the vulnerable software from a previous epoch off System A and apply it to System B in order to attack it. The fact that the vulnerable software from an older epoch was personalized to System A prevents it from being transferable and thus being used to attack a System B. All these mechanisms work together to provide much stronger guarantees that attackers can’t purposely place vulnerable software on a computer in order to circumvent the protections provided by the latest software. But a user who is in possession of an administrator user name and password for the Mac can still always choose the security policy that works best for their use cases.

Medium Security boot policy

Medium Security is somewhat like a traditional UEFI secure boot situation, where a vendor (here, Apple) generates a digital signature for the code to assert it came from the vendor. In this way attackers are prevented from inserting unsigned code. We refer to this signature as a “global” signature, because it can be used on any Mac, for any amount of time, for those Mac computers that currently have a Medium Security policy set. Neither iOS, iPadOS, nor the T2 chip support global signatures.

A limitation of global signature schemes has to do with the prevention of “rollback attacks.” In a rollback attack, an attacker places old, but legitimate and correctly signed, software with known vulnerabilities onto a system and then exploits those vulnerabilities to take control of the system. Many global signature systems don’t attempt to prevent rollback attacks at all. Those that do often do this through the use of a “security version” or “security epoch.” This is a number that is typically covered by the signature, and evaluated after the signature has been verified. The computer needs secure persistent storage to keep track of the largest epoch value it has ever seen in signed code, and to disallow any code—even if it is properly signed—that has an epoch less than this.

A vendor wanting to roll the epoch signs software with a new epoch, one that is greater than any previously issued software contained. Firmware detecting an epoch value greater than the latest observed one in its secure storage updates the value of the epoch in the storage. It then subsequently rejects all previous signed code with epochs less than the latest stored value. If the system doesn't have secure storage, an attacker can simply roll back the epoch value itself and can then roll back and exploit the software. This is why many systems that do implement epochs store the epoch number in a one-time-programmable fuse array. When the fuses are burned out, the values can't be changed. However, this also has the limitation that an attacker can simply burn all the fuses in order to render all signatures invalid, thereby permanently preventing the operating system to boot.

The Apple global signature scheme doesn't include a security epoch, because those systems are inflexible and frequently cause significant usability issues. Protection against rollback attacks is better achieved by Full Security mode, which is the default, and very similar in behavior to iOS and iPadOS. Users who want to take advantage of anti-rollback protection should retain the default Full Security policy. However, the Medium Security mode is made available for those users who may not be able to take advantage of Full Security mode.

Media boot policy

Media boot policy is only shown on Mac computers with an Apple T2 Security Chip and is completely independent from the secure boot policy. Even if a user disables secure boot, this doesn't change the default behavior of disallowing boot from anything other than the storage device directly connected to the T2 chip.

Historically, Mac computers have been able to boot from an external device by default. This approach would allow an attacker with physical possession of the device to run arbitrary code from the booted volume. The combination of protections like FileVault and SecureBoot make it so that there are no known architectural weaknesses through which an attacker running from an external volume can access the user's data without knowledge of that user's password. However, having even temporary arbitrary code execution can allow an attacker to manipulate the Mac in ways that can stage attacker-controlled data to exploit vulnerabilities that are unknown to Apple. Arbitrary code creation can thus possibly lead to a user boot being compromised and to subsequent user data compromise.

Apple changed the policy for external boot to default-deny, and to opt out on Mac computers with a T2 chip. On Mac computers without a T2 chip, users could always set a firmware password to opt in to this default-deny behavior. However, firmware passwords were not well known and received very little adoption. With this policy change, Apple is changing the behavior of the Mac to give the best protection possible by default, rather than putting the onus on users to opt in.

Firmware Password protection

macOS supports the use of a Firmware Password to prevent unintended modifications of firmware settings on a specific Mac. The Firmware Password is used to prevent selecting alternate boot modes such as booting into recoveryOS or Single-User mode, booting from an unauthorized volume, or booting into Target Disk Mode.

The most basic mode of Firmware Password can be reached from the recoveryOS Firmware Password Utility on Mac computers without an Apple T2 Security Chip, and from the Startup Security Utility on Mac computers with a T2 chip. Advanced options (such as the ability to prompt for the password at every boot) are available from the `firmwarepasswd` command-line tool in macOS.

As described in Boot process of Mac computers without an Apple T2 Security Chip, setting a Firmware Password is especially important to reduce the risk of attacks on Mac computers without a T2 chip via a physically present attacker (such as in a computer lab or office environment). The firmware password can stop an attacker from booting to recoveryOS, from where they can disable System Integrity Protection. And by restricting boot of alternative media, an attacker can't execute privileged code from another OS in order to attack peripheral firmwares.

A Firmware Password reset mechanism exists to help users who forget their password. Users press a key combination at boot, and be presented with a model-specific string to provide to an AppleCare. AppleCare digitally signs a resource that is signature-checked by the Uniform Resource Identifier (URI). If the signature validates and the content is for the specific Mac, the UEFI firmware removes the Firmware Password.

For users who want no one but themselves to remove their Firmware Password by software means, the `–disable–reset–capability` option has been added to the `firmwarepasswd` command-line tool in macOS 10.15. Before setting this option, users must to acknowledge that if the password is forgotten and needs removal, the user must bear the cost of the motherboard replacement necessary to achieve this. Organizations that want to protect their Mac computers from external attackers and from employees must set a Firmware Password on organization-owned systems. This can be accomplished on the device:

- At provisioning time, by manually using the `firmwarepasswd` command-line tool
- With third-party management tools that use the `firmwarepasswd` command-line tool
- Using mobile device management (MDM)

Secure software updates

Secure software updates overview

Apple regularly releases software updates to address emerging security concerns and to provide new features; these updates are generally provided for all supported devices simultaneously. Users of iOS and iPadOS devices receive update notifications on the device and through iTunes (in macOS 10.14 or earlier) or the Finder (macOS 10.15 or later). macOS updates are available in System Preferences. Updates are delivered wirelessly, for rapid adoption of the latest security fixes.

The startup process helps ensure that only Apple-signed code is being installed. For example, System Software Authorization ensures that only legitimate copies of operating system versions that are actively being signed by Apple can be installed on iOS and iPadOS devices, or Mac computers with the Full Security setting configured as the secure boot policy in the Startup Security Utility. This system prevents iOS and iPadOS devices from being downgraded to older versions that lack the latest security updates, and can be used by Apple to prevent similar downgrades in macOS. Without this protection, an attacker who gains possession of a device could install an older version of iOS or iPadOS and exploit a vulnerability that's been fixed in newer versions.

In addition, when a device is physically connected to a Mac, a full copy of iOS or iPadOS is downloaded and installed. But for over-the-air (OTA) software updates, only the components required to complete an update are downloaded, improving network efficiency by not downloading the entire OS. Additionally, software updates can be cached on a Mac running macOS 10.13 or later with Content Caching turned on, so that iOS and iPadOS devices don’t need to redownload the necessary update over the Internet. They’ll still need to contact Apple servers to complete the update process.

Secure software update process

During upgrades, a connection is made to the Apple installation authorization server, which includes a list of cryptographic measurements for each part of the installation bundle to be installed (for example, iBoot, the kernel, and OS image), a random anti-replay value (nonce), and the device’s unique Exclusive Chip Identification (ECID).

The authorization server checks the presented list of measurements against versions for which installation is permitted and, if it finds a match, adds the ECID to the measurement and signs the result. The server passes a complete set of signed data to the device as part of the upgrade process. Adding the ECID “personalizes” the authorization for the requesting device. By authorizing and signing only for known measurements, the server ensures that the update takes place exactly as Apple provided.

The boot-time chain-of-trust evaluation verifies that the signature comes from Apple and that the measurement of the item loaded from the storage device, combined with the device’s ECID, matches what was covered by the signature. These steps ensure that the authorization is for a specific device and that an older iOS, iPadOS, or Apple T2 Security Chip’s firmware version from one device can’t be copied to another. The nonce prevents an attacker from saving the server’s response and using it to tamper with a device or otherwise alter the system software.

On a device with Secure Enclave, the Secure Enclave coprocessor also uses System Software Authorization to ensure the integrity of its software and prevent downgrade installations.

OS integrity in iOS and iPadOS

iOS and iPadOS system security overview

Apple designed the iOS platform with security at its core. When we set out to create the best possible mobile platform, we drew from decades of experience to build an entirely new architecture. We thought about the security hazards of the desktop environment and established a new approach to security in the design of iOS. We developed and incorporated innovative features that tighten mobile security and protect the entire system by default. As a result, iOS and subsequently iPadOS are a major leap forward in security for mobile devices.

Kernel Integrity Protection

After the iOS and iPadOS kernels complete initialization, Kernel Integrity Protection (KIP) is enabled to prevent modifications of kernel and driver code. The memory controller provides a protected physical memory region that iBoot uses to load the kernel and kernel extensions. After boot completes, the memory controller denies writes to the protected physical memory region. Additionally, the application processor's Memory Management Unit (MMU) is configured to prevent mapping privileged code from physical memory outside the protected memory region, and to prevent writeable mappings of physical memory within the kernel memory region.

To prevent reconfiguration, the hardware used to enable KIP is locked after the boot process is complete. KIP is supported on SoCs starting with the Apple A10 and S4.

For the Apple A11 Bionic SoC, a new hardware primitive has been introduced. This primitive includes a CPU register that quickly restricts permissions per thread. With these fast permission restrictions (or APRR), iOS and iPadOS are able to remove execute permissions from memory—without the overhead of a system call and a page table walk or flush.

System Coprocessor Integrity Protection

Coprocessor firmware handles many critical system tasks—for example, the Secure Enclave, the image sensor processor, and the Motion coprocessor. Therefore its security is a key part of the security of the overall system. To prevent modification of coprocessor firmware, Apple uses a mechanism called System Coprocessor Integrity Protection (SCIP), supported on SoCs starting with the Apple A12 and S4 SoCs.

SCIP works much like Kernel Integrity Protection: At boot time, iBoot loads each coprocessor's firmware into a protected memory region, one that's reserved and separate form the KIP region. iBoot configures each coprocessor's memory unit to prevent:

- Executable mappings outside its part of the protected memory region
- Writeable mappings inside its part of the protected memory region

Also at boot time, to configure SCIP for the Secure Enclave, the Secure Enclave operating system is used. After the boot process is complete, the hardware used to enable SCIP is locked to prevent reconfiguration.

Pointer Authentication Codes

Pointer authentication codes (PACs) are supported starting with the Apple A12 and S4 SoCs and used to protect against exploitation of memory corruption bugs. System software and built-in apps use PAC to prevent modification of function pointers and return addresses (code pointers). PAC uses five secret 128-bit values to sign kernel instructions and data, and each user space process has its own B keys. Items are salted and signed as indicated below:

Item	Key	Salt
Function Return Address	IB	Storage address
Function Pointers	IA	0
Block Invocation Function	IA	Storage address
Objective-C Method Cache	IB	Storage address + Class + Selector
C++ V-Table Entries	IA	Storage address + Hash (mangled method name)
Computed Goto Label	IA	Hash (function name)
Kernel Thread State	GA	•
User Thread State Registers	IA	Storage address
C++ V-Table Pointers	DA	0

The signature value is stored in the unused padding bits at the top of the 64-bit pointer. The signature is verified before use, and the padding is restored to ensure a functioning pointer address. Failure to verify results in a special value being set which invalidates the address, and in iOS 13 and iPadOS 13.1 it aborts. This verification increases the difficulty of many attacks, such as a Return Oriented Programming (ROP) attack, which attempts to trick the device into executing existing code maliciously by manipulating function return addresses stored on the stack. PACs are supported starting with the Apple A12 and S4 SoCs.

Page Protection Layer

Page Protection Layer (PPL) in iOS and iPadOS protects userland code from modification after code signature verification completes. It builds on KIP and APRR to carefully manage the page table permission overrides to make sure only the PPL can alter protected pages containing user code and page tables. The system provides a massive reduction in attack surface by supporting systemwide code integrity enforcement, even in the face of a compromised kernel.

OS integrity in macOS

macOS system security overview

Apple designed the macOS platform with an integrated approach to hardware, software, and services that provides security by design and makes it simple to configure, deploy, and manage. macOS includes the key security technologies that an IT professional needs to help protect corporate data and integrate within secure enterprise networking environments. Apple has also worked with standards bodies to ensure compliance with the latest security certifications.

Mac firmware security

UEFI firmware security overview

Since 2006, Mac computers with an Intel-based CPU use an Intel firmware based on the Extensible Firmware Interface (EFI) Development Kit (EDK) version 1 or version 2. EDK2-based code conforms to the Unified Extensible Firmware Interface (UEFI) specification. This section refers to the Intel firmware as the UEFI Firmware. The UEFI firmware was the first code to execute on the Intel chip.

In order to prevent attacks that physically attach to the firmware storage chip that stores UEFI firmware, Mac computers were rearchitected starting in 2017 to root the trust in the UEFI firmware stored in the Apple T2 Security Chip. On these Mac computers, the root of trust for the UEFI firmware is specifically the T2 firmware. This design relies on the T2 to protect the UEFI firmware (and Secure Boot as a whole) from persistent infection, in the much the same way that boot is protected by the A Series SoC's in iOS and iPadOS.

For Mac computers without the Apple T2 Security Chip, the root of trust for the UEFI firmware is the chip where the firmware is stored. UEFI firmware updates are digitally signed by Apple and verified by the firmware before updating the storage. To prevent rollback attacks, updates must always have a version newer than the existing one. However, an attacker with physical access to the Mac could use hardware to attach to the firmware storage chip and update the chip to contain malicious content. Likewise, if vulnerabilities are found in the early boot process of the UEFI firmware (before it write-restricts the storage chip) this could also lead to persistent infection of the UEFI firmware. This is a hardware architectural limitation common in most Intel-based PCs which is present in all Mac computers without the T2 chip.

To address this limitation, Mac computers were rearchitected to root the trust in the UEFI firmware in the Apple T2 Security Chip. On these Mac computers, the root of trust for the UEFI firmware is specifically the T2 firmware, as described in the macOS boot section later in this section. To achieve a persistent UEFI firmware infection, an attacker would need to achieve a persistent T2 firmware infection.

Intel Management Engine (ME)

One subcomponent which is stored within the UEFI firmware is the Intel Management Engine (ME) firmware. The ME—a separate processor and subsystem within Intel chips—can be used for remote management, protected audio and video, and security enhancement. To reduce that attack surface, Mac computers run a custom ME firmware from which the majority of components have been removed. This allows the Mac ME firmware to be smaller than the default minimal build that Intel makes available. Consequently, many components (such as Active Management Technology) that have been the subject of public attacks by security researchers in the past are not present within Mac ME firmware. The primary use of the ME is audio and video copyright protection on Mac computers that have only Intel-based graphics.

System Management Mode (SMM)

Intel processors have a special execution mode that is distinct from normal operation. Called System Management Mode (SMM), it was originally introduced to handle time-sensitive operations such as power management. However, to perform such actions, Mac computers have historically used a discrete microcontroller called the System Management Controller (SMC). The SMC is no longer a separate microcontroller, it has been integrated into the Apple T2 Security Chip.

On PCs that support secure boot, SMM serves an additional role as being a protected execution environment that can be given exclusive access to security-sensitive content such as write access to the code and security policy stored on the UEFI firmware storage chip. As such, it's often in an attacker's interest to break into the SMM execution environment as a form of privilege escalation, to perform operations that a kernel can't and thus potentially compromise secure boot. On Mac computers, the SMM execution environment is used as little as possible, and isn't treated as a security boundary for secure boot purposes. Therefore even if SMM is compromised, Secure Boot is unaffected. On the T2 chip, the privilege boundary is instead the action that the chip can perform exclusively.

DMA protections

To achieve high throughput on high-speed interfaces like PCIe, FireWire, Thunderbolt, and USB, computers must support Direct Memory Access (DMA) from peripherals. That is, they must be able to read and write to RAM without continuous involvement of the Intel CPU. Since 2102, Mac computers have implemented numerous technologies to protect DMA, resulting in the best and most comprehensive set of DMA protections on any PC.

Intel Virtualization Technology for Directed IO (VT-d) is a technology which has been supported since 2012 on Mac computers, and was first used in OS X 10.9 in order to protect the kernel from being overwritten in memory by malicious peripherals. However, malicious peripherals can also overwrite code and data *while* the UEFI firmware is running in order to compromise boot security. macOS 10.12.3 updated the UEFI firmware for all VT-d-capable Mac computers to use VT-d to protect against malicious FireWire and Thunderbolt peripherals. It also isolates peripherals so that they can see only their own memory ranges, not the memory of other peripherals. For example, an Ethernet peripheral running in UEFI can't read the memory of a storage peripheral.

DMA protections in UEFI firmware were further improved in macOS 10.13 to move the initialization earlier in the UEFI firmware startup sequence to protect against:

- Malicious internal peripheral processors on the PCIe bus
- A class of Message Signaled Interrupt (MSI) attacks presented by security researchers

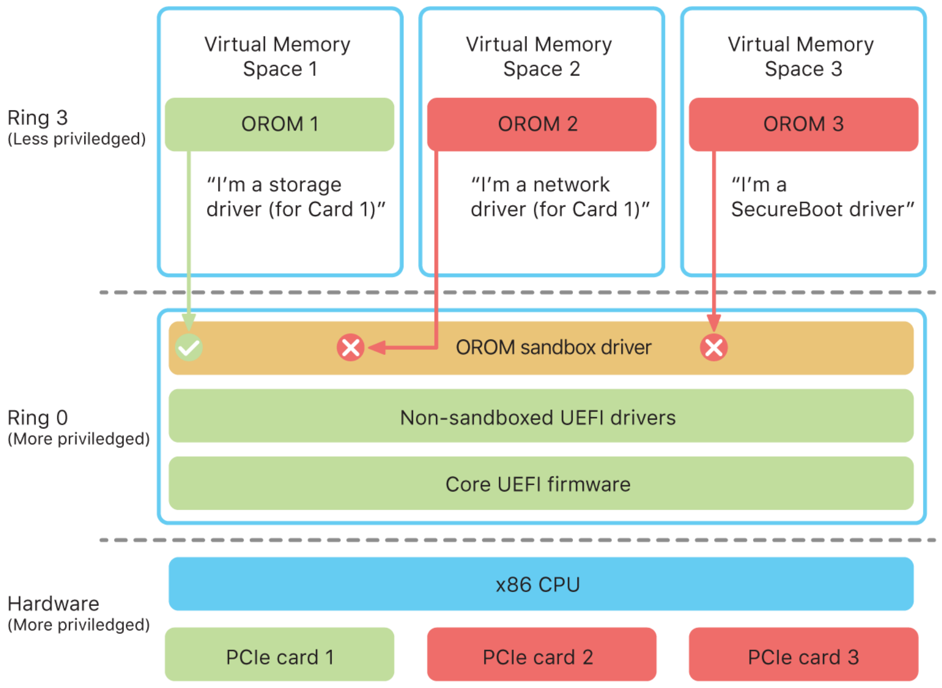
All Mac computers with the Apple T2 Security Chip come with further improved DMA protections, where the initialization is performed as early as possible. Specifically, the protection is enabled before any RAM is even available to the UEFI firmware. This protects against any compromised PCIe bus zero devices (such as the Intel ME) that may be running and capable of DMA at the instant that RAM becomes available. This protection was also added to Mac computers without a T2 chip in macOS 10.15.

Option ROMs

Both Thunderbolt and PCIe devices can have an "Option ROM" (OROM) physically attached to the device. (This is typically not a true ROM, but instead a rewritable chip that stores firmware.) On UEFI-based systems, that firmware is typically a UEFI driver, which is read in by the UEFI firmware and executed. The executed code is supposed to initialize and configure the hardware it was retrieved from, so that the hardware can be made usable by the rest of the firmware. This capability is required so that specialized third-party hardware can load and operate during the earliest boot phases—for example, to boot from external RAID arrays.

However, because OROMs are generally rewritable, if an attacker overwrites the OROM of a legitimate peripheral, the attacker's code would execute early in the boot process, and be able to tamper with the execution environment and violate the integrity of subsequently loaded software. Likewise, if the attacker introduces their own malicious device to the system, they would also be able to execute malicious code.

In macOS 10.12.3, the behavior of Mac computers sold after 2011 was changed to not execute OROMs by default at the time the Mac booted, unless a special key combination was pressed. This key combination protected against malicious OROMs being inadvertently introduced into the macOS boot sequence. The default behavior of the Firmware Password Utility was also changed so that when the user set a firmware password, OROMs couldn't execute even *if* the key combination was pressed. This protected against a physically present attacker intentionally introducing a malicious OROM. For users who still need to run OROMs while they have a firmware password set, a non default option can be configured using the `firmwarepasswd` command-line tool in macOS.



Option ROM (OROM) sandboxing.

OROM sandbox

In macOS 10.15, UEFI firmware was updated to contain a mechanism for sandboxing and deprivileging OROMs. UEFI firmware typically executes all code, including OROMs, at the maximum Intel CPU privilege level, called ring 0, and a single shared virtual memory space for all code and data. Ring 0 is also the privilege level at which the macOS kernel runs, whereas the lower privilege level, ring 3, is where apps run. The OROM sandbox deprivileged OROMs by making use of virtual memory separation like the kernel does, and then making the OROMs run in ring 3.

The sandbox further significantly restricts both the interfaces that the OROMs can call (which is similar to system call filtering in kernels), and the type of device that an OROM can register as (which is similar to app whitelisting.) The benefit of this design is that malicious OROMs can no longer directly write anywhere within ring 0 memory, and are instead limited to a very narrow and well-defined sandbox interface. This limited interface significantly reduces attack surface and forces attackers to first escape the sandbox and escalate privilege.

Peripheral firmware security

Mac computers have many built-in peripheral processors dedicated to tasks such as networking, graphics, power management, or managing data buses like USB or Thunderbolt. Often peripheral firmware is single-purpose, and much less powerful than the Intel CPU. However, built-in peripherals that don't implement sufficient security become a target for attackers seeking even easier targets to exploit and then persistently infect the operating system. Having infected a peripheral processor firmware, an attacker could target software on the Intel CPU, or directly capture sensitive data (for example, an Ethernet device could see the contents of packets which aren't encrypted).

Apple strategically works with third-party vendors to reduce (wherever possible) the number of peripheral processors necessary, or to avoid designs that require firmware. But when firmware is required, efforts are taken to ensure an attacker can't persist on that processor. This can be achieved:

- By running the processor in a mode where it downloads verified firmware from the Intel CPU on startup
- By ensuring the peripheral processor implements its own secure boot chain where it verifies its own firmware on every boot

Apple works with vendors to audit their implementations, and enhance their designs to include desired properties such as:

- Ensuring minimum cryptographic strengths
- Strong revocation of known bad firmware
- Disabling debug interfaces
- Signing the firmware with cryptographic keys that are stored in Apple-controlled Hardware Security Modules (HSMs)

In recent years Apple has worked with some external vendors to adopt the same "Image4" data structures, verification code, and signing infrastructure used by iOS, iPadOS, and Mac computers with the Apple T2 Security Chip.

When neither storage-free operation nor storage plus secure boot is an option, the design mandates that firmware updates be cryptographically signed and verified before the persistent storage can be updated.

Mandatory access controls

macOS also uses mandatory access controls—policies that set security restrictions, created by the developer, that can't be overridden. This approach is different from discretionary access controls, which permit users to override security policies according to their preferences.

Mandatory access controls aren't visible to users, but they're the underlying technology that helps enable several important features, including sandboxing, parental controls, managed preferences, extensions, and System Integrity Protection.

System Integrity Protection

OS X 10.11 or later includes system-level protection, called *System Integrity Protection*, which restricts components to read-only in specific critical file system locations to prevent malicious code from modifying them. System Integrity Protection is a computer-specific setting that's on by default when a user upgrades to OS X 10.11 or later; disabling it removes protection for all partitions on the physical storage device. macOS applies this security policy to every process running on the system, regardless of whether it's running sandboxed or with administrative privileges.

Kernel extensions

Kernel extensions (KEXTs) are no longer recommended for macOS. KEXTs risk the integrity and reliability of the operating system, and users should prefer solutions that don’t require extending the kernel.

macOS 10.15 supports the ability for developers to extend the capabilities of macOS by installing and managing system extensions that run in user space rather than at the kernel level. By running in user space, system extensions increase the stability and security of macOS. While KEXTs inherently have full access to the entire operating system, extensions running in user space are granted only the privileges necessary to perform their specified function.

Developers can use frameworks including DriverKit, EndpointSecurity, and NetworkExtension to write USB and human interface drivers, endpoint security tools (like data loss prevention or other endpoint agents), and VPN and network tools—all without needing to write KEXTs. Third-party security agents should be used only if they take advantage of these APIs or have a robust roadmap to transition to them and away from kernel extensions.

macOS still provides a kernel extension mechanism to allow dynamic loading of code into the kernel without the need to recompile or relink. To improve security, user consent is required to load kernel extensions installed with or after installing macOS 10.13. This is known as User-Approved Kernel Extension Loading. Administrator authorization is required to approve a kernel extension.

Kernel extensions don’t require authorization if they:

- Were installed on the Mac before upgrading to macOS 10.13
- Are replacing previously approved extensions
- Are allowed to load without user consent by using the `spctl` command-line tool available when booted from macOS Recovery
- Are allowed to load using mobile device management (MDM) configuration

  Starting with macOS 10.13.2, users can use MDM to specify a list of kernel extensions that load without user consent. This option requires a Mac running macOS 10.13.2 that’s enrolled in MDM—through Apple School Manager, Apple Business Manager, or user-approved MDM enrollment.

System security in watchOS

watchOS system security overview

Apple Watch uses the security features and technology built for iOS and iPadOS to help protect data on the device, and to protect communication with its paired iPhone and with the Internet. This includes technologies such as Data Protection and Keychain access control. The user's passcode is also entangled with the device UID to create encryption keys.

Pairing Apple Watch with iPhone is secured using an out-of-band (OOB) process to exchange public keys, followed by the Bluetooth Low Energy (BLE) link shared secret. Apple Watch displays an animated pattern, which is captured by the camera on iPhone. The pattern contains an encoded secret that is used for BLE 4.1 out-of-band pairing. Standard BLE Passkey Entry is used as a fallback pairing method, if necessary.

Once the BLE session is established and encrypted using the highest security protocol available in Bluetooth Core Specification, Apple Watch and iPhone exchange keys using a process adapted from Apple Identity Service (IDS), as described iMessage overview. After keys have been exchanged, the Bluetooth session key is discarded and all communications between Apple Watch and iPhone are encrypted using IDS, with the encrypted Bluetooth, Wi-Fi, and Cellular links providing a secondary encryption layer. The Low Energy Bluetooth Address is rotated at 15-minute intervals to reduce the risk of local tracking of the device using the broadcast of a persistent identifier.

To support apps that need streaming data, encryption is provided using methods described in FaceTime, utilizing either the IDS service provided by the paired iPhone or a direct Internet connection.

Apple Watch implements hardware-encrypted storage and class-based protection of files and Keychain items, as described in the Encryption and Data Protection section of this paper. Access-controlled keybags for Keychain items are also used. Keys used for communications between the watch and iPhone are also secured using class-based protection.

When Apple Watch isn't within Bluetooth range, Wi-Fi or cellular can be used instead. Apple Watch automatically joins Wi-Fi networks that have been already been joined on the paired iPhone and whose credentials have synced to the Apple Watch while both devices were in range. This Auto-Join behavior can then be configured on a per network basis in the Wi-Fi section of the Apple Watch Settings app. Wi-Fi networks that have never been joined before on either device can be manually joined in Wi-Fi section of the Apple Watch Settings app.

When Apple Watch and iPhone are out of range, Apple Watch connects directly to iCloud and Gmail servers to fetch Mail, as opposed to syncing Mail data with the paired iPhone over the Internet. For Gmail accounts, the user is required to authenticate to Google in the Mail section of the Watch app on iPhone. The OAuth token received from Google is sent over to Apple Watch in encrypted format over Apple Identity Service (IDS) so it can be used to fetch Mail. This OAuth token is never used for connectivity with the Gmail server from the paired iPhone.

Apple Watch can be manually locked by holding down the side button. Additionally, unless wrist detection is disabled, the device locks automatically shortly after it's removed from the user's wrist. When Apple Watch is locked, Apple Pay can only be used by entering the watch's passcode. Wrist detection is turned off using the Apple Watch app on iPhone. This setting can also be enforced using a mobile device management (MDM) solution.

The paired iPhone can also unlock the watch, provided the watch is being worn. This is accomplished by establishing a connection authenticated by the keys established during pairing. iPhone sends the key, which the watch uses to unlock its Data Protection keys. The watch passcode isn’t known to iPhone nor is it transmitted. This feature can be turned off using the Apple Watch app on iPhone.

Apple Watch can be paired with only one iPhone at a time. iPhone communicates instructions to erase all content and data from Apple Watch when unpaired.

Apple Watch can be configured for a system software update the same night. For more information on how the Apple Watch passcode gets stored to be used during the update see Keybags in iOS and iPadOS.

Enabling Find My on the paired iPhone also allows the use of Activation Lock on Apple Watch. Activation Lock makes it harder for anyone to use or sell an Apple Watch that has been lost or stolen. Activation Lock requires the user’s Apple ID and password to unpair, erase, or reactivate an Apple Watch.

Apple Watch usage with macOS

Auto Unlock with Apple Watch in macOS

Users with Apple Watch can use it to automatically unlock their Mac. Bluetooth Low Energy (BLE) and peer-to-peer Wi-Fi allow Apple Watch to securely unlock a Mac after ensuring proximity between the devices. This requires an iCloud account with two-factor authentication (TFA) configured.

When enabling an Apple Watch to unlock a Mac, a secure link using Auto Unlock Identities is established. The Mac creates a random one-time-use unlock secret and transmits it to the Apple Watch over the link. The secret is stored on Apple Watch and can only be accessed when Apple Watch is unlocked. The unlock token is not the user’s password.

During an unlock operation, the Mac uses BLE to create a connection to the Apple Watch. A secure link is then established between the two devices using the shared keys used when it was first enabled. The Mac and Apple Watch then use peer-to-peer Wi-Fi and a secure key derived from the secure link to determine the distance between the two devices. If the devices are within range, the secure link is then used to transfer the preshared secret to unlock the Mac. After successful unlock, the Mac replaces the current unlock secret with a new one-time use unlock secret and transmits the new unlock secret to the Apple Watch over the link.

Approve with Apple Watch

When Auto Unlock with Apple Watch is enabled, the Apple Watch can be used in place or together with Touch ID to approve authorization and authentication prompts from:

- macOS and Apple apps that request authorization
- Third-party apps that request authentication
- Saved Safari passwords
- Secure Notes

Encryption and Data Protection

Encryption and Data Protection overview

The secure boot chain, system security, and app security capabilities all help to ensure that only trusted code and apps run on a device. Apple devices have additional encryption features to safeguard user data, even when other parts of the security infrastructure have been compromised (for example, if a device is lost or is running untrusted code). All of these features benefit both users and IT administrators, protecting personal and corporate information at all times and providing methods for instant and complete remote wipe in the case of device theft or loss.

iOS and iPadOS devices use a file encryption methodology called Data Protection, while the data on Mac computers is protected with a volume encryption technology called FileVault. Both models similarly root their key management hierarchies in the dedicated silicon of the Secure Enclave (on devices that include a SEP), and both models leverage a dedicated AES engine to support line-speed encryption and to ensure that long-lived encryption keys never need to be provided to the kernel OS or CPU (where they might be compromised).

How Apple protects users' personal information

In addition to encrypting data at rest, Apple devices help prevent apps from accessing a user's personal information without permission. In Settings in iOS, iPadOS, or System Preferences in macOS, users can see which apps they have permitted to access certain information, as well as grant or revoke any future access. Access is enforced in the following:

- *iOS, iPadOS, and macOS:* Calendars, Camera, Contacts, Microphone, Photos, Reminders, Speech recognition
- *iOS and iPadOS:* Bluetooth, Home, Media, Media apps and Apple Music, Motion and fitness
- *iOS and watchOS:* Health
- *macOS:* Input monitoring (for example, keyboard strokes), Prompt, Screen recording (for example, static screen shots and video), System Preferences

If the user signs in to iCloud, apps in iOS and iPadOS are granted access by default to iCloud Drive. Users may control each app's access under iCloud in Settings. Additionally, iOS and iPadOS provide restrictions that prevent data movement between apps and accounts installed by a mobile device management (MDM) solution and those installed by the user.

Role of Apple File System

Apple File System (APFS) is a proprietary file system that was designed with encryption in mind. APFS works across all Apple's platforms—for iOS, iPadOS, macOS, tvOS, and watchOS. Optimized for Flash/SSD storage, it features strong encryption, copy-on-write metadata, space sharing, cloning for files and directories, snapshots, fast directory sizing, atomic safe-save primitives, and improved file system fundamentals, as well as a unique copy-on-write design that uses I/O coalescing to deliver maximum performance while ensuring data reliability.

APFS allocates storage space on demand. When a single APFS container has multiple volumes, the container's free space is shared and can be allocated to any of the individual volumes as needed. Each volume uses only part of the overall container, so the available space is the total size of the container, minus the space used in all volumes in the container.

In macOS 10.15 an APFS container used to start up the Mac must contain at least five volumes, the first three of which are hidden from the user:

- *Preboot volume:* Contains data needed for booting each system volume in the container
- *VM volume:* Used by macOS for swap file storage
- *Recovery volume:* Contains recoveryOS
- *System volume:* Contains the following:
  - All the necessary files to start up the Mac
  - All apps installed natively by macOS (apps that used to reside in the /Applications folder now reside in /System/Applications)
- *Data volume:* Contains data that is subject to change, such as:
  - Any data inside the user's folder, including photos, music, videos, and documents
  - Apps the user installed, including AppleScript, and Automator applications
  - Custom frameworks and daemons installed by the user, organization, or third-party apps
  - Other locations owned and writable by the user, as /Applications, /Library, /Users, /Volumes, /usr/local, /private, /var, and /tmp

A Data volume is created for each additional System volume. The Preboot, VM, and Recovery volume are all shared and not duplicated.

Note: By default, no process can write to the System volume, even Apple system processes.

Data Protection in iOS and iPadOS

Data Protection overview

In iOS and iPadOS, Apple uses a technology called Data Protection to protect data stored in flash storage on the device. Data Protection allows the device to respond to common events such as incoming phone calls, but also enables a high level of encryption for user data. Key system apps, such as Messages, Mail, Calendar, Contacts, Photos, and Health data values use Data Protection by default, and third-party apps installed on iOS 7 or later and iPadOS 13.1 receive this protection automatically.

Implementation

Data Protection is implemented by constructing and managing a hierarchy of keys, and builds on the hardware encryption technologies built into each iOS and iPadOS device. Data Protection is controlled on a per-file basis by assigning each file to a class; accessibility is determined according to whether the class keys have been unlocked. With the advent of the Apple File System (APFS), the file system is now able to further subdivide the keys into a per-extent basis (where portions of a file can have different keys).

Architecture

In iOS and iPadOS, storage is divided into two APFS volumes:

- *System volume:* System content is stored on the System volume, and user data is stored on the Data volume.
- *Data volume:* Every time a file on the data volume is created, Data Protection creates a new 256-bit key (the per-file key) and gives it to the hardware AES engine, which uses the key to encrypt the file as it is written to flash storage. The encryption uses AES128 in XTS mode where the 256-bit per file key is split to provide a 128-bit tweak and a 128-bit cipher key.

How data files are created and protected

On devices with an A7, S2, or S3 SoC, AES-CBC is used. The initialization vector is calculated with the block offset into the file, encrypted with the SHA-1 hash of the per-file key.

The per-file (or per-extent) key is wrapped with one of several class keys, depending on the circumstances under which the file should be accessible. Like all other wrappings that use RFC 3394, this is performed using NIST AES key wrapping. The wrapped per-file key is stored in the file's metadata.

Devices with APFS format may support cloning of files (zero-cost copies using copy-on-write technology). If a file is cloned, each half of the clone gets a new key to accept incoming writes so that new data is written to the media with a new key. Over time, the file may become composed of various extents (or fragments), each mapping to different keys. However, all of the extents that comprise a file are guarded by the same class key.

When a file is opened, its metadata is decrypted with the file system key, revealing the wrapped per-file key and a notation on which class protects it. The per-file (or per-extent) key is unwrapped with the class key and then supplied to the hardware AES engine, which decrypts the file as it's read from flash storage. All wrapped file key handling occurs in the Secure Enclave; the file key is never directly exposed to the Intel CPU. At boot time, the Secure Enclave negotiates an ephemeral key with the AES engine. When the Secure Enclave unwraps a file's keys, they are rewrapped with the ephemeral key and sent back to the application processor.

The metadata of all files in the data volume file system are encrypted with a random volume key, which is created when iOS and iPadOS are first installed or when the device is wiped by a user. This key is encrypted and wrapped by a key wrapping key that is known only to the Secure Enclave for long term storage. The key wrapping key changes every time a user erases their device. On A9 (and newer) SoCs, Secure Enclave relies upon entropy, backed by anti-replay nonce, to achieve effaceability and to protect its key wrapping key, among other assets.

Just like per-file or per-extent keys, the metadata key of the data volume is never directly exposed to the application processor; the Secure Enclave provides an ephemeral, per-boot version instead. When stored, the encrypted file system key is additionally wrapped by an "effaceable key" stored in Effaceable Storage. This key doesn't provide additional confidentiality of data. Instead, it's designed to be quickly erased on demand (by the user with the "Erase All Content and Settings" option, or by a user or administrator issuing a remote wipe command from a mobile device management (MDM) solution, Microsoft Exchange ActiveSync, or iCloud). Erasing the key in this manner renders all files cryptographically inaccessible.

The contents of a file may be encrypted with one or more per-file (or per-extent) keys that are wrapped with a class key and stored in a file's metadata, which in turn is encrypted with the file system key. The class key is protected with the hardware UID and, for some classes, the user's passcode. This hierarchy provides both flexibility and performance. For example, changing a file's class only requires rewrapping its per-file key, and a change of passcode just rewraps the class key.

Data Protection classes

When a new file is created on an iOS or iPadOS device, it's assigned a class by the app that creates it. Each class uses different policies to determine when the data is accessible. The basic classes and policies are described in the following sections.

Complete Protection

(NSFileProtectionComplete): The class key is protected with a key derived from the user passcode and the device UID. Shortly after the user locks a device (10 seconds, if the Require Password setting is Immediately), the decrypted class key is discarded, rendering all data in this class inaccessible until the user enters the passcode again or unlocks the device using Touch ID or Face ID.

Protected Unless Open

(NSFileProtectionCompleteUnlessOpen): Some files may need to be written while the device is locked. A good example of this is a mail attachment downloading in the background. This behavior is achieved by using asymmetric elliptic curve cryptography (ECDH over Curve25519). The usual per-file key is protected by a key derived using One-Pass Diffie-Hellman Key Agreement as described in NIST SP 800-56A.

The ephemeral public key for the Agreement is stored alongside the wrapped per-file key. The KDF is Concatenation Key Derivation Function (Approved Alternative 1) as described in 5.8.1 of NIST SP 800-56A. AlgorithmID is omitted. PartyUInfo and PartyVInfo are the ephemeral and static public keys, respectively. SHA-256 is used as the hashing function. As soon as the file is closed, the per-file key is wiped from memory. To open the file again, the shared secret is re-created using the Protected Unless Open class's private key and the file's ephemeral public key, which are used to unwrap the per-file key that is then used to decrypt the file.

Protected Until First User Authentication

(NSFileProtectionCompleteUntilFirstUserAuthentication): This class behaves in the same way as Complete Protection, except that the decrypted class key isn't removed from memory when the device is locked. The protection in this class has similar properties to desktop full-volume encryption, and protects data from attacks that involve a reboot. This is the default class for all third-party app data not otherwise assigned to a Data Protection class.

No Protection

(NSFileProtectionNone): This class key is protected only with the UID, and is kept in Effaceable Storage. Since all the keys needed to decrypt files in this class are stored on the device, the encryption only affords the benefit of fast remote wipe. If a file isn't assigned a Data Protection class, it is still stored in encrypted form (as is all data on an iOS and iPadOS device).

Data Protection class key

Class	**Protection type**
Class A: Complete Protection	(NSFileProtectionComplete)
Class B: Protected Unless Open	(NSFileProtectionCompleteUnlessOpen)
Class C: Protected Until First User Authentication	(NSFileProtectionCompleteUntilFirstUserAuthentication)
Class D: No Protection	(NSFileProtectionNone)

Accessing protected keys in recovery modes

On devices with Apple A10, A11, and S3 SoCs, class keys protected by the user's passcode can't be accessed from Recovery mode. The A12 and S4 SoCs extend this protection to Device Firmware Upgrade (DFU) mode.

The Secure Enclave AES engine is equipped with lockable software seed bits. When keys are created from the UID, these seed bits are included in the key derivation function to create additional key hierarchies.

Starting with the Apple A10 and S3 SoCs, a seed bit is dedicated to distinguish keys protected by the user's passcode. The seed bit is set for keys that require the user's passcode (including Data Protection Class A, Class B, and Class C keys), and cleared for keys that don't require the user's passcode (including the filesystem metadata key and Class D keys).

On A12 SoCs, the Secure Enclave Boot ROM locks the passcode seed bit if the application processor has entered DFU mode or Recovery mode. When the passcode seed bit is locked, no operation to change it is allowed, preventing access to data protected with the user's passcode.

On Apple A10, A11, S3, and S4 SoCs, the passcode seed bit is locked by the Secure Enclave OS if the device has entered Recovery mode. The Secure Enclave Boot ROM and OS both check the Boot Progress Register (BPR) to securely determine the current mode.

In addition, in iOS 13 and iPadOS 13.1 or later on devices with an A10 or newer, all user data is rendered cryptographically inaccessible when devices are booted into Recovery mode. This is achieved by introducing an additional seed bit whose setting governs the ability to access the media key, which itself is needed to access the metadata (and therefore contents of) all files on the data volume encrypted with Data Protection. This protection encompasses files protected in all classes (A, B, C, and D), not just those that required the user's passcode.

Restoring a device after it enters DFU mode returns it to a known good state with the certainty that only unmodified Apple-signed code is present. DFU mode can be entered manually.

See the following Apple Support articles on how to place a device in DFU mode:

Device	Article
iPhone, iPad, iPod touch	If you forgot the passcode for your iPhone, iPad, or iPod touch, or your device is disabled
Apple TV	Restore your Apple TV

Keychain data protection and data classes

Keychain data protection overview

Many apps need to handle passwords and other short but sensitive bits of data, such as keys and login tokens. The iOS and iPadOS Keychain provides a secure way to store these items.

Keychain items are encrypted using two different AES-256-GCM keys: a table key (metadata), and a per-row key (secret-key). Keychain metadata (all attributes other than kSecValue) is encrypted with the metadata key to speed searches while the secret value (kSecValueData) is encrypted with the secret-key. The meta-data key is protected by the Secure Enclave, but is cached in the application processor to allow fast queries of the keychain. The secret key always requires a roundtrip through the Secure Enclave.

The Keychain is implemented as a SQLite database, stored on the file system. There is only one database and the `securityd` daemon determines which Keychain items each process or app can access. Keychain access APIs result in calls to the daemon, which queries the app's "Keychain-access-groups," "application-identifier," and "application-group" entitlements. Rather than limiting access to a single process, access groups allow Keychain items to be shared between apps.

Keychain items can only be shared between apps from the same developer. This is managed by requiring third-party apps to use access groups with a prefix allocated to them through the Apple Developer Program through application groups. The prefix requirement and application group uniqueness are enforced through code signing, provisioning profiles, and the Apple Developer Program.

Keychain data is protected using a class structure similar to the one used in file Data Protection. These classes have behaviors equivalent to file Data Protection classes but use distinct keys and are part of APIs that are named differently.

Availability	**File Data Protection**	**Keychain Data Protection**
When unlocked	NSFileProtectionComplete	kSecAttrAccessibleWhenUnlocked
While locked	NSFileProtectionCompleteUnless Open	N/A
After first unlock	NSFileProtectionCompleteUntil FirstUserAuthentication	kSecAttrAccessibleAfterFirstUnloc k
Always	NSFileProtectionNone	kSecAttrAccessibleAlways
Passcode enabled	N/A	kSecAttrAccessibleWhenPasscode SetThisDeviceOnly

Apps that utilize background refresh services can use *kSecAttrAccessibleAfterFirstUnlock* for Keychain items that need to be accessed during background updates.

The class *kSecAttrAccessibleWhenPasscodeSetThisDeviceOnly* behaves the same as *kSecAttrAccessibleWhenUnlocked*; however, it's available only when the device is configured with a passcode. This class exists only in the system Keybag; they:

- Don't sync to iCloud Keychain
- Aren't backed up
- Aren't included in escrow keybags

If the passcode is removed or reset, the items are rendered useless by discarding the class keys.

Other Keychain classes have a "This device only" counterpart, which is always protected with the UID when being copied from the device during a backup, rendering it useless if restored to a different device. Apple has carefully balanced security and usability by choosing Keychain classes that depend on the type of information being secured and when it's needed by iOS and iPadOS. For example, a VPN certificate must always be available so the device keeps a continuous connection, but it's classified as "non-migratory," so it can't be moved to another device.

Keychain data class protections

For Keychain items created by iOS and iPadOS, the following class protections are enforced:

Item	Accessible
Wi-Fi passwords	After first unlock
Mail accounts	After first unlock
Microsoft Exchange ActiveSync accounts	After first unlock
VPN passwords	After first unlock
LDAP, CalDAV, CardDAV	After first unlock
Social network account tokens	After first unlock
Handoff advertisement encryption keys	After first unlock
iCloud token	After first unlock
Home sharing password	When unlocked
Safari passwords	When unlocked
Safari bookmarks	When unlocked
iTunes backup	When unlocked, non-migratory
VPN certificates	Always, non-migratory
Bluetooth® keys	Always, non-migratory
Apple Push Notification service (APNs) token	Always, non-migratory
iCloud certificates and private key	Always, non-migratory
iMessage keys	Always, non-migratory
Certificates and private keys installed by a configuration profile	Always, non-migratory
SIM PIN	Always, non-migratory
Find My token	Always
Voicemail	Always

Keychain access control

Keychains can use access control lists (ACLs) to set policies for accessibility and authentication requirements. Items can establish conditions that require user presence by specifying that they can't be accessed unless authenticated using Touch ID, Face ID, or by entering the device's passcode. Access to items can also be limited by specifying that Touch ID or Face ID enrollment hasn't changed since the item was added. This limitation helps prevent an attacker from adding their own fingerprint in order to access a Keychain item. ACLs are evaluated inside the Secure Enclave and are released to the kernel only if their specified constraints are met.

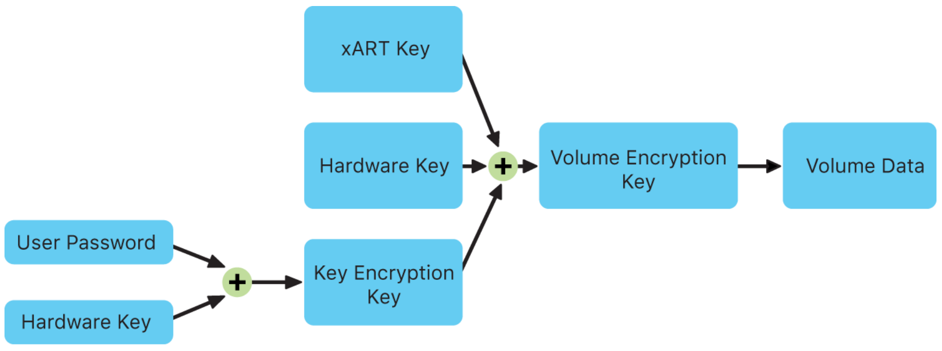
Encryption in macOS

Internal volume encryption when FileVault is turned on

In Mac OS X 10.3 or later, Mac computers provide FileVault, a built-in encryption capability to secure all data at rest. FileVault uses the AES-XTS data encryption algorithm to protect full volumes on internal and removable storage devices. On Mac computers with the Apple T2 Security Chip, encrypted internal storage devices directly connected to the T2 chip leverage the hardware security capabilities of the chip. After a user turns on FileVault on a Mac, their credentials are required during the boot process.

Without valid login credentials or a cryptographic recovery key, the internal APFS volume (in macOS 10.15, this includes the System and Data volumes) remains encrypted and is protected from unauthorized access even if the physical storage device is removed and connected to another computer. Internal volume encryption on a Mac with the T2 chip is implemented by constructing and managing a hierarchy of keys, and builds on the hardware encryption technologies built into the chip. This hierarchy of keys is designed to simultaneously achieve four goals:

- Require the user's password for decryption
- Protect the system from a brute-force attack directly against storage media removed from Mac
- Provide a swift and secure method for wiping content via deletion of necessary cryptographic material
- Enable users to change their password (and in turn the cryptographic keys used to protect their files) without requiring reencryption of the entire volume



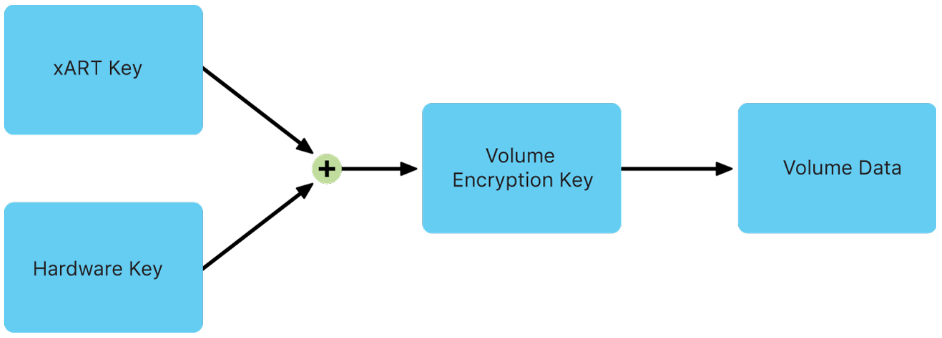
Internal volume encryption when FileVault is turned on in macOS.

On Mac computers with the T2 chip, all FileVault key handling occurs in the Secure Enclave; encryption keys are never directly exposed to the Intel CPU. All APFS volumes are created with a volume key by default. Volume and metadata contents are encrypted with this volume key, which is wrapped with the class key. The class key is protected by a combination of the user's password and the hardware UID when FileVault is turned on. This protection is the default on Mac computers with the T2 chip.

Note: Encryption of removable storage devices doesn't utilize the security capabilities of the Apple T2 Security Chip, and its encryption is performed in the same manner as Mac computers without the T2 chip.

Internal volume encryption when FileVault is turned off

If FileVault isn't turned on on a Mac with the Apple T2 Security Chip during the initial Setup Assistant process, the volume is still encrypted but the volume key is protected only by the hardware UID in the Secure Enclave.



Internal volume encryption when FileVault is turned off in macOS.

If FileVault is turned on later—a process that is immediate since the data was already encrypted—an anti-replay mechanism prevents the old key (based on hardware UID only) from being used to decrypt the volume. The volume is then protected by a combination of the user password with the hardware UID as previously described.

Deleting FileVault volumes

When deleting a volume, its volume key is securely deleted by Secure Enclave. This prevents future access with this key even by the Secure Enclave. In addition, all volume keys are wrapped with a media key. The media key doesn’t provide additional confidentiality of data, but instead is designed to enable swift and secure deletion of data because without it, decryption is impossible.

The media key is located in Effaceable Storage and designed to be quickly erased on demand—for example, using remote wipe using Find My or when enrolled in a mobile device management (MDM) solution. Effaceable Storage accesses the underlying storage technology (for example, NAND) to directly address and erase a small number of blocks at a very low level. Erasing the media key in this manner renders the volume cryptographically inaccessible.

Preventing brute force attacks and malware

To prevent brute-force attacks, when Mac boots, no more than 30 password attempts are allowed at the Login Window or using Target Disk Mode, and escalating time delays are imposed after incorrect attempts. The delays are enforced by the Secure Enclave coprocessor on the T2 chip. If Mac is restarted during a timed delay, the delay is still enforced, with the timer starting over for the current period.

To prevent malware from causing permanent data loss by trying to attack the user’s password, these limits are not enforced after the user has successfully logged into the Mac, but is reimposed after reboot. If the 30 attempts are exhausted, 10 more attempts are available after booting into macOS Recovery. And if those are also exhausted, then 60 additional attempts are available for each FileVault recovery mechanism (iCloud recovery, FileVault recovery key, and institutional key), for a maximum of 180 additional attempts. Once those additional attempts are exhausted, the Secure Enclave no longer processes any requests to decrypt the volume or verify the password, and the data on the drive becomes unrecoverable.

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.

Delays between password attempts

Attempts	**Delay enforced**
1–14	None
15-17	1 minute
18-20	5 minutes
21-26	15 minutes
27-30	1 hour

Managing FileVault

Using SecureToken

Apple File System (APFS) in macOS 10.13 or later changes how FileVault encryption keys are generated. In previous versions of macOS on CoreStorage volumes, the keys used in the FileVault encryption process were created when a user or organization turned on FileVault on a Mac. In macOS on APFS volumes, the keys are generated either during user creation or during the first login by a user of the Mac. This implementation of the encryption keys, when they are generated, and how they are stored are a part of SecureToken. Specifically, a SecureToken is a wrapped version of a Key Encryption Key (KEK) protected by a user's password.

When deploying FileVault on APFS, the user can continue to:

- Use existing tools and processes, such as Personal Recovery Key (PRK) escrow to a mobile device management (MDM) solution
- Create and use an Institutional Recovery Key (IRK)
- Defer enablement of FileVault until a user logs in to or out of the Mac

Using Bootstrap Token

macOS 10.15 introduces a new feature—Bootstrap Token—to help with granting a SecureToken to both mobile accounts and the optional device enrollment-created administrator account ("managed administrator"). The managed administrator can be created by configuring an MDM solution to create it during the enrollment process that happens using Apple School Manager or Apple Business Manager. Using the new Bootstrap Token feature of macOS 10.15 requires:

- Mac enrollment in MDM using Apple School Manager or Apple Business Manager
- MDM vendor support

Note: A Bootstrap Token can't be generated automatically by macOS during setup if the first user created in Setup Assistant is downgraded to a standard user using MDM or if local user account creation is skipped entirely. If needed, the initially created administrator account in Setup Assistant can be downgraded to a standard user later, or can be abandoned or deleted if not needed (It can't be deleted or downgraded until at least one other Secure Token–enabled administrator account exists on the Mac). Alternatively, a Bootstrap Token can be generated after macOS setup using the profiles command-line tool.

When a user sets up a Mac on their own

When a user sets up a Mac on their own, IT departments don't provision the actual device. All policies and configurations are provided using a mobile device management (MDM) solution or configuration management tools. Setup Assistant is used to create the initial local administrator account, and the user is granted a SecureToken. If the MDM solution supports the Bootstrap Token feature and informs the Mac during MDM enrollment, a Bootstrap Token is generated by the Mac and escrowed to the MDM solution.

If the Mac is enrolled in an MDM solution, the initial account may not be a local administrator account but rather a local standard user account. If the user is downgraded to a standard user using MDM, the user is automatically granted a SecureToken. If the user is downgraded, no Bootstrap Token is generated.

Note: If local user account creation in Setup Assistant is skipped altogether using MDM and a directory service with mobile accounts is used instead, the directory user won't be granted a SecureToken during login and no Bootstrap Token is generated. If there are no SecureToken users on the Mac, the mobile account can still be enabled for FileVault using deferred enablement and SecureToken is granted to the user at the time that FileVault is turned on.

In any of the above scenarios, because the first and primary user is granted a SecureToken, they can be enabled for FileVault using deferred enablement. Deferred enablement allows the organization to turn on FileVault, but defer its enablement until a user logs into or out of the Mac. It's also possible to customize if the user can skip turning on FileVault (optionally a defined number of times). The end result is the primary user of the Mac—whether a local user of any type or a mobile account—being able to unlock the storage device when encrypted with FileVault.

On Mac computers where a Bootstrap Token was generated and escrowed to an MDM solution, if the managed administrator account logs in to the Mac at a future date and time, the Bootstrap Token is used to automatically grant a SecureToken, meaning the account is also enabled for FileVault and able to unlock the FileVault volume. To modify whether the managed administrator account can unlock the volume, the user can use: `fdesetup remove —user`.

When a Mac is provisioned by an organization

When a Mac is provisioned by an organization before being given to a user, the IT department sets up the device. The local administrative account created in the macOS Setup Assistant used to provision or set up the Mac is granted a SecureToken. In macOS 10.15, if the MDM solution supports the Bootstrap Token feature, a Bootstrap Token may also be generated during the macOS setup process and escrowed to the MDM solution. If the managed administrator account logs in to the Mac at a future date and time, the Bootstrap Token is used to automatically grant it SecureToken.

If the Mac is joined to a directory service and configured to create mobile accounts and if there is no Bootstrap Token, directory service users are prompted at first login for an existing SecureToken administrator's user name and password to grant their account a SecureToken. The local administrator credentials used to set up the Mac should be entered. If SecureToken isn't required, the user clicks Bypass. In macOS 10.13.5 or later, it's possible to suppress the SecureToken dialog completely if FileVault isn't going to be used with the mobile accounts. To suppress the SecureToken dialog, apply a custom settings configuration profile from MDM with the following keys and values:

Setting	Value
Domain	com.apple.MCX
Key	cachedaccounts.askForSecureTokenAuthBypass
Value	True

If the MDM solution supports the Bootstrap Token feature and one was generated by the Mac during setup and escrowed to the MDM solution, Mobile Account users won't see this prompt. Instead, they are automatically granted a SecureToken during login.

If additional local users are required on the Mac instead of using a directory service, those local users are automatically granted a SecureToken when they are created in System Preferences > Users & Groups by a current SecureToken-enabled administrator.

In these scenarios, the following users can unlock the FileVault-encrypted volume:

- The original local administrator used for provisioning
- Any additional directory service users granted SecureToken during the login process, either interactively using the dialog prompt or automatically with the Bootstrap Token
- Any new local users created in System Preferences

To modify the whether specific accounts can unlock the storage device, the user can use`fdesetup remove -user`.

When using one of the above described workflows, SecureToken is managed by macOS without any additional configuration or scripting being needed; it becomes an implementation detail and not something that needs to be actively managed or manipulated.

Using command-line tools

Command-line tools are available for managing Bootstrap Token, FileVault, and SecureToken. The Bootstrap Token is usually generated on the Mac and escrowed to the mobile device management (MDM) solution during the macOS setup process after the MDM solution tells the Mac that it supports the feature. However, a Bootstrap Token can also be generated on a Mac that has already been deployed. For example, if the MDM solution adds support for this feature after an initial deployment of macOS 10.15. The `profiles` command-line tool has a number of new options to interact with the Bootstrap Token.

Commands to manage the Bootstrap token:

- `sudo profiles install -type bootstraptoken`: This command generates a new Bootstrap Token and attempts to escrow it to the MDM solution. This command requires existing SecureToken administrator information to initially generate the Bootstrap Token, the MDM solution must support the feature, and the Mac must be enrolled in MDM and Apple School Manager or Apple Business Manager.

- `sudo profiles remove –type bootstraptoken`: Removes the existing Bootstrap Token on the Mac and the MDM solution.
- `sudo profiles status –type bootstraptoken`: Reports back whether the MDM solution supports the Bootstrap Token feature, and what the current state of the Bootstrap Token is on the Mac.
- `sudo profiles validate –type bootstraptoken`: Verifies that the Bootstrap Token escrowed in the MDM solution is valid on the Mac.

fdesetup command-line tool

MDM configurations or the `fdesetup` command-line tool can be used to configure FileVault. In macOS 10.15 or later, using `fdesetup` to turn on FileVault by providing the user name and password is deprecated and won't be recognized in a future release. Consider using deferred enablement using MDM instead. To learn more about the `fdesetup` command-line tool, launch the Terminal app and enter `man fdesetup` or `fdesetup help` for additional information.

sysadminctl command-line tool

The `sysadminctl` command-line tool can be used to specifically to modify SecureToken status for user accounts on the Mac. This should be done with caution and only when necessary. Changing the SecureToken status of a user using `sysadminctl` always requires the user name and password of an existing SecureToken-enabled administrator, either interactively or through the appropriate flags on the command. Both `sysadminctl` and System Preferences prevent the deletion of the last administrator or SecureToken-enabled user on a Mac. If the creation of additional local users is scripted using `sysadminctl`, for those users to be enabled for SecureToken, current SecureToken-enabled administrator credentials are required to be supplied either using the interactive option or directly with the `–adminUser` and `–adminPassword` flags with `sysadminctl`. Use `sysadminctl –h` for additional usage instructions.

Passcodes and passwords

Passcodes

By setting up a device passcode, the user automatically enables Data Protection. iOS and iPadOS support six-digit, four-digit, and arbitrary-length alphanumeric passcodes. In addition to unlocking the device, a passcode provides entropy for certain encryption keys. This means an attacker in possession of a device can't get access to data in specific protection classes without the passcode.

The passcode is entangled with the device's UID, so brute-force attempts must be performed on the device under attack. A large iteration count is used to make each attempt slower. The iteration count is calibrated so that one attempt takes approximately 80 milliseconds. This means it would take more than five and one-half years to try all combinations of a six-character alphanumeric passcode with lowercase letters and numbers.

The stronger the user passcode is, the stronger the encryption key becomes. Touch ID and Face ID can be used to enhance this equation by enabling the user to establish a much stronger passcode than would otherwise be practical. This increases the effective amount of entropy protecting the encryption keys used for Data Protection, without adversely affecting the user experience of unlocking an iOS or iPadOS device multiple times throughout the day.

To further discourage brute-force passcode attacks, there are escalating time delays after the entry of an invalid passcode at the Lock screen. If Settings > Touch ID& Passcode > Erase Data is turned on, the device automatically wipes after 10 consecutive incorrect attempts to enter the passcode. Consecutive attempts of the same incorrect passcode don't count toward the limit. This setting is also available as an administrative policy through a mobile device management (MDM) solution that supports this feature and Microsoft Exchange ActiveSync, and can be set to a lower threshold.

On devices with Secure Enclave, the delays are enforced by the Secure Enclave coprocessor. If the device is restarted during a timed delay, the delay is still enforced, with the timer starting over for the current period.

Specifying longer passcodes

If a long password that contains only numbers is entered, a numeric keypad is displayed at the Lock screen instead of the full keyboard. A longer numeric passcode may be easier to enter than a shorter alphanumeric passcode, while providing similar security.

Users can specify a longer alphanumeric passcode by selecting Custom Alphanumeric Code in the Passcode Options in Settings > Passcode.

Delays between passcode attempts

Attempts	**Delay enforced**
1–4	None
5	1 minute
6	5 minutes
7–8	15 minutes
9	1 hour

Activating data connections securely

To improve security while maintaining usability Touch ID, Face ID, or passcode entry is required to activate data connections via the Lightning, USB, or Smart Connector interface if no data connection has been established recently. This limits the attack surface against physically connected devices such as malicious chargers while still enabling usage of other accessories within reasonable time constraints. If more than an hour has passed since the iOS or iPadOS device has locked or since an accessory's data connection has been terminated, the device won't allow any new data connections to be established until the device is unlocked. During this hour period, only data connections from accessories that have been previously connected to the device while in an unlocked state will be allowed. These accessories are remembered for 30 days after the last time they were connected. Attempts by an unknown accessory to open a data connection during this period will disable all accessory data connections over Lighting, USB, and Smart Connector until the device is unlocked again. This hour period:

- Ensures that frequent users of connections to a Mac or PC, to accessories, or wired to CarPlay won't need to input their passcodes every time they attach their device.
- Is necessary because the accessory ecosystem doesn't provide a cryptographically reliable way to identify accessories before establishing a data connection.

In addition, if it's been more than three days since a data connection has been established with an accessory, the device will disallow new data connections immediately after it locks. This is to increase protection for users that don't often make use of such accessories. Data connections over Lightning, USB, and Smart Connector are also disabled whenever the device is in a state where it requires a passcode to reenable biometric authentication.

The user can choose to reenable always-on data connections in Settings (setting up some assistive devices does this automatically).

Function of passwords

In Mac computers with the Apple T2 Security Chip, the password serves a similar function to passcodes above, except that the key generated is used for FileVault encryption rather than data protection. macOS also offers additional password recovery options:

- iCloud recovery
- FileVault recovery
- FileVault institutional key

Authentication and digital signing

Digital signing and encryption

Access Control Lists

Keychain data is partitioned and protected with Access Control Lists (ACLs). As a result, credentials stored by third-party apps can't be accessed by apps with different identities unless the user explicitly approves them. This protection provides the mechanism for securing authentication credentials in Apple devices across a range of apps and services within the organization.

Mail

In the Mail app, users can send messages that are digitally signed and encrypted. Mail automatically discovers appropriate RFC 5322 case-sensitive email address subject or subject alternative names on digital signing and encryption certificates on attached PIV tokens in compatible smart cards. If a configured email account matches an email address on a digital signing or encryption certificate on an attached PIV token, Mail automatically displays the signing button in the toolbar of a new message window. If Mail has the recipient's email encryption certificate or can discover it in the Microsoft Exchange Global Address List (GAL), an unlocked icon appears in the new message toolbar. A locked lock icon indicates the message will be sent encrypted with the recipient's public key.

Per-message S/MIME

iOS, iPadOS, and macOS support per-message S/MIME. This means that S/MIME users can choose to always sign and encrypt messages by default or to selectively sign and encrypt individual messages.

Identities used with S/MIME can be delivered to Apple devices using a configuration profile, a mobile device management (MDM) solution, the Simple Certificate Enrollment Protocol (SCEP), or Microsoft Active Directory Certificate Authority.

Smart cards

macOS 10.12 or later includes native support for personal identity verification (PIV) cards. These cards are widely used in commercial and government organizations for TFA, digital signing, and encryption.

Smart cards include one or more digital identities that have a pair of public and private keys and an associated certificate. Unlocking a smart card with the personal identification number (PIN) provides access to the private keys used for authentication, encryption, and signing operations. The certificate determines what a key can be used for, what attributes are associated with it, and whether it's validated (signed) by a CA.

Smart cards can be used for two-factor authentication. The two factors needed to unlock a card are "something the user has" (the card) and "something the user knows" (the PIN). macOS 10.12 or later also has native support for smart card login window authentication and client certificate authentication to websites on Safari. It also supports Kerberos authentication using key pairs (PKINIT) for single sign-on to Kerberos-supported services. To learn more about Smart cards and macOS, see Intro to smart card integration in the *Deployment Reference for Mac*.

Encrypted disk images

In macOS, encrypted disk images serve as secure containers in which users can store or transfer sensitive documents and other files. Encrypted disk images are created using Disk Utility, located in /Applications/Utilities/. Disk images can be encrypted using either 128-bit or 256-bit AES encryption. Because a mounted disk image is treated as a local volume connected to a Mac, users can copy, move, and open files and folders stored in it. As with FileVault, the contents of a disk image are encrypted and decrypted in real time. With encrypted disk images, users can safely exchange documents, files, and folders by saving an encrypted disk image to removable media, sending it as a mail message attachment, or storing it on a remote server. For more information on encrypted disk images, see the Disk Utility User Guide.

Keychain architecture in macOS

macOS offers a repository, called Keychain, that conveniently and securely stores user names and passwords, including digital identities, encryption keys, and secure notes. It can be accessed by opening the Keychain Access app in /Applications/Utilities/. Using a keychain eliminates the requirement to enter—or even remember—the credentials for each resource. An initial default keychain is created for each Mac user, though users can create other keychains for specific purposes.

In addition to user keychains, macOS relies on a number of system-level keychains that maintain authentication assets that aren't user-specific, such as network credentials and public key infrastructure (PKI) identities. One of these keychains, System Roots, is immutable and stores Internet PKI root certificate authority (CA) certificates to facilitate common tasks like online banking and e-commerce. The user can similarly deploy internally provisioned CA certificates to managed Mac computers to help validate internal sites and services.

Keybags

Keybags overview in iOS and iPadOS

The keys for both file and Keychain Data Protection classes are collected and managed in keybags. iOS and iPadOS use the following keybags: user, device, backup, escrow, and iCloud Backup.

User keybag

The user keybag is where the wrapped class keys used in normal operation of the device are stored. For example, when a passcode is entered, *NSFileProtectionComplete* is loaded from the user keybag and unwrapped. It is a binary property list (.plist) file stored in the No Protection class.

For devices with SoCs earlier than the A9, the .plist file contents are encrypted with a key held in Effaceable Storage. In order to give forward security to keybags, this key is wiped and regenerated each time a user changes their passcode.

For devices with the A9 or newer SoCs, the .plist file contains a key that indicates that the keybag is stored in a locker protected by Secure Enclave controlled anti-replay nonce.

The Secure Enclave manages the user keybag and can be queried regarding a device's lock state. It reports that the device is unlocked only if all the class keys in the user keybag are accessible and have been unwrapped successfully.

Device keybag

The device keybag is used to store the wrapped class keys used for operations involving device-specific data. iOS and iPadOS devices configured for shared use sometimes need access to credentials before any user has logged in; therefore, a keybag that isn't protected by the user's passcode is required.

iOS and iPadOS don't support cryptographic separation of per-user file system content, which means the system uses class keys from the device keybag to wrap per-file keys. The Keychain, however, uses class keys from the user keybag to protect items in the user Keychain. In iOS and iPadOS devices configured for use by a single user (the default configuration), the device keybag and the user keybag are one and the same, and are protected by the user's passcode.

Backup keybag

The backup keybag is created when an encrypted backup is made by iTunes (inmacOS 10.14 or earlier) or the Finder (macOS 10.15 or later) and stored on the computer to which the device is backed up. A new keybag is created with a new set of keys, and the backed-up data is reencrypted to these new keys. As explained previously, non-migratory Keychain items remain wrapped with the UID-derived key, allowing them to be restored to the device they were originally backed up from but rendering them inaccessible on a different device.

The keybag—protected with the password set in iTunes (in macOS 10.14 or earlier) or the Finder (macOS 10.15 or later)—is run through 10 million iterations of PBKDF2. Despite this large iteration count, there's no tie to a specific device, and therefore a brute-force attack parallelized across many computers could theoretically be attempted on the backup keybag. This threat can be mitigated with a sufficiently strong password.

If a user chooses not to encrypt the backup, the files aren't encrypted regardless of their Data Protection class but the Keychain remains protected with a UID-derived key. This is why Keychain items migrate to a new device only if a backup password is set.

Escrow keybag

The escrow keybag is used for iTunes syncing and mobile device management (MDM). This keybag allows iTunes to back up and sync without requiring the user to enter a passcode, and it allows an MDM solution to remotely clear a user's passcode. It is stored on the computer that's used to sync with iTunes, or on the MDM solution that remotely manages the device.

The escrow keybag improves the user experience during device synchronization, which potentially requires access to all classes of data. When a passcode-locked device is first connected to iTunes, the user is prompted to enter a passcode. The device then creates an escrow keybag containing the same class keys used on the device, protected by a newly generated key. The escrow keybag and the key protecting it are split between the device and the host or server, with the data stored on the device in the Protected Until First User Authentication class. This is why the device passcode must be entered before the user backs up with iTunes for the first time after a reboot.

In the case of an over-the-air (OTA) software update, the user is prompted for their passcode when initiating the update. This is used to securely create a one-time Unlock Token, which unlocks the user keybag after the update. This token can't be generated without entering the user's passcode, and any previously generated token is invalidated if the user's passcode changed.

One-time Unlock Tokens are either for attended or unattended installation of a software update. They are encrypted with a key derived from the current value of a monotonic counter in the Secure Enclave, the UUID of the keybag, and the Secure Enclave's UID.

For devices with SoCs earlier than the A9, incrementing the one-time Unlock Token counter in the Secure Enclave invalidates any existing token. The counter is incremented when a token is used, after the first unlock of a restarted device, when a software update is canceled (by the user or by the system), or when the policy timer for a token has expired.

On A9 (and newer) SoCs, one-time Unlock token no longer relies on counters or Effaceable Storage. Instead, it's protected by Secure Enclave controlled anti-replay nonce.

The one-time Unlock Token for attended software updates expires after 20 minutes. Prior to iOS 13, this token is exported from the Secure Enclave and is written to Effaceable Storage. A policy timer increments the counter if the device hasn't rebooted within 20 minutes. In iOS 13 and iPadOS 13.1, the token is stored in a locker protected by the Secure Enclave.

Unattended software updates occur when the system detects an update is available and:

- Automatic updates are configured in iOS 12 (or later)

or

- The user chooses "Install Later" when notified of the update

After the user enters their passcode, a one-time Unlock Token is generated and can remain valid in Secure Enclave for up to 8 hours. If the update hasn't yet occurred, this one-time Unlock Token is destroyed on every lock and recreated on every subsequent unlock. Each unlock restarts the 8 hour window. After 8 hours a policy timer invalidates the one-time Unlock Token.

iCloud Backup keybag

The iCloud Backup keybag is similar to the backup keybag. All the class keys in this keybag are asymmetric (using Curve25519, like the Protected Unless Open Data Protection class). An asymmetric keybag is also used for the backup in the Keychain recovery aspect of iCloud Keychain.

App Security

App security overview

Apps are among the most critical elements of a modern security architecture. While apps provide amazing productivity benefits for users, they also have the potential to negatively impact system security, stability, and user data if they’re not handled properly.

Because of this, Apple provides layers of protection to ensure that apps are free of known malware and haven’t been tampered with. Additional protections enforce that access from apps to user data is carefully mediated. These security controls provide a stable, secure platform for apps, enabling thousands of developers to deliver hundreds of thousands of apps for iOS, iPadOS, and macOS—all without impacting system integrity. And users can access these apps on their Apple devices without undue fear of viruses, malware, or unauthorized attacks.

On iPhone, iPad, and iPod touch, all apps are obtained from the App Store—and all apps are sandboxed—to provide the tightest controls.

On Mac, many apps are obtained from the App Store, but Mac users also download and use apps from the Internet. To safely support Internet downloading, macOS layers additional controls. First, by default on macOS 10.15 or later, all Mac apps need to be notarized by Apple to launch. This requirement ensures that these apps are free of known malware without requiring that the apps be provided through the App Store. In addition, macOS includes industry-standard anti-virus protection to block—and if necessary remove—malware.

As an additional control across platforms, sandboxing helps protect user data from unauthorized access by apps. And in macOS, data in critical areas is itself sandboxed—which ensures—that users remain in control of access to files in Desktop, Documents, Downloads, and other areas from all apps, whether the apps attempting access are themselves sandboxed or not.

Native capability	Third-party equivalent
Plug-in unapproved list, Safari extension unapproved list	Virus/Malware Definitions
File quarantine	Virus/Malware Definitions
XProtect/Yara signatures	Virus/Malware Definitions
MRT (Malware Removal Tool)	Endpoint protection
Gatekeeper	Endpoint protection. Enforces code signing on apps to ensure only trusted software runs.
eficheck (Necessary for Mac computers without an Apple T2 Security chip)	Endpoint protection–rootkit detection
Application firewall	Endpoint protection–firewalling
ipfw	Firewall solutions
System Integrity Protection	Only Apple can provide this
Mandatory Access Controls	Only Apple can provide this
KEXT exclude list	Only Apple can provide this
Mandatory app code signing	Only Apple can provide this
App notarization	Only Apple can provide this

App security in iOS and iPadOS

iOS and iPadOS app security overview

Unlike other mobile platforms, iOS and iPadOS don’t allow users to install potentially malicious unsigned apps from websites, or run untrusted apps. At runtime, code signature checks of all executable memory pages are made as they are loaded to ensure that an app hasn’t been modified since it was installed or last updated.

After an app is verified to be from an approved source, iOS and iPadOS enforce security measures designed to prevent it from compromising other apps or the rest of the system.

App code signing process

Mandatory code signing

After the iOS or iPadOS kernel has started, it controls which user processes and apps can be run. To ensure that all apps come from a known and approved source and haven’t been tampered with, iOS and iPadOS require that all executable code be signed using an Apple-issued certificate. Apps provided with the device, like Mail and Safari, are signed by Apple. Third-party apps must also be validated and signed using an Apple-issued certificate. Mandatory code signing extends the concept of chain of trust from the OS to apps and prevents third-party apps from loading unsigned code resources or using self-modifying code.

How developers sign their apps

Certificate validation

To develop and install apps in iOS or iPadOS devices, developers must register with Apple and join the Apple Developer Program. The real-world identity of each developer, whether an individual or a business, is verified by Apple before their certificate is issued. This certificate enables developers to sign apps and submit them to the App Store for distribution. As a result, all apps in the App Store have been submitted by an identifiable person or organization, serving as a deterrent to the creation of malicious apps. They have also been reviewed by Apple to ensure they generally operate as described and don’t contain obvious bugs or other notable problems. In addition to the technology already discussed, this curation process gives users confidence in the quality of the apps they buy.

Validation of dynamic libraries

iOS and iPadOS allow developers to embed frameworks inside of their apps, which can be used by the app itself or by extensions embedded within the app. To protect the system and other apps from loading third-party code inside of their address space, the system performs a code signature validation of all the dynamic libraries that a process links against at launch time. This verification is accomplished through the team identifier (Team ID), which is extracted from an Apple-issued certificate. A team identifier is a 10-character alphanumeric string—for example, 1A2B3C4D5F. A program may link against any platform library that ships with the system or any library with the same team identifier in its code signature as the main executable. Since the executables shipping as part of the system don’t have a team identifier, they can only link against libraries that ship with the system itself.

Verifying enterprise apps

Businesses also have the ability to write in-house apps for use within their organization and distribute them to their employees. Businesses and organizations can apply to the Apple Developer Enterprise Program (ADEP) with a D-U-N-S number. Apple approves applicants after verifying their identity and eligibility. After an organization becomes a member of ADEP, it can register to obtain a provisioning profile that permits in-house apps to run on devices it authorizes.

Users must have the provisioning profile installed to run the in-house apps. This ensures that only the organization’s intended users are able to load the apps onto their iOS and iPadOS devices. Apps installed through mobile device management (MDM) are implicitly trusted because the relationship between the organization and the device is already established. Otherwise, users have to approve the app's provisioning profile in Settings. Organizations can restrict users from approving apps from unknown developers. On first launch of any enterprise app, the device must receive positive confirmation from Apple that the app is allowed to run.

Security of runtime process

Sandboxing

All third-party apps are "sandboxed," so they are restricted from accessing files stored by other apps or from making changes to the device. Sandboxing prevents apps from gathering or modifying information stored by other apps. Each app has a unique home directory for its files, which is randomly assigned when the app is installed. If a third-party app needs to access information other than its own, it does so only by using services explicitly provided by iOS and iPadOS.

System files and resources are also shielded from the user's apps. The majority of iOS and iPadOS run as the nonprivileged user "mobile," as do all third-party apps. The entire OS partition is mounted as read-only. Unnecessary tools, such as remote login services, aren't included in the system software, and APIs don't allow apps to escalate their own privileges to modify other apps or iOS and iPadOS.

Use of entitlements

Access by third-party apps to user information, and to features such as iCloud and extensibility, is controlled using declared entitlements. Entitlements are key-value pairs that are signed in to an app and allow authentication beyond runtime factors, like UNIX user ID. Since entitlements are digitally signed, they can't be changed. Entitlements are used extensively by system apps and daemons to perform specific privileged operations that would otherwise require the process to run as root. This greatly reduces the potential for privilege escalation by a compromised system app or daemon.

In addition, apps can only perform background processing through system-provided APIs. This enables apps to continue to function without degrading performance or dramatically impacting battery life.

Further protections

Address Space Layout Randomization

Address Space Layout Randomization (ASLR) protects against the exploitation of memory corruption bugs. Built-in apps use ASLR to ensure that all memory regions are randomized upon launch. Randomly arranging the memory addresses of executable code, system libraries, and related programming constructs reduces the likelihood of many sophisticated exploits. For example, a return-to-libc attack attempts to trick a device into executing malicious code by manipulating memory addresses of the stack and system libraries. Randomizing the placement of these makes the attack far more difficult to execute, especially across multiple devices. Xcode, the iOS or iPadOS development environments, automatically compiles third-party programs with ASLR support turned on.

Execute Never

Further protection is provided by iOS and iPadOS using ARM's Execute Never (XN) feature, which marks memory pages as non-executable. Memory pages marked as both writable and executable can be used only by apps under tightly controlled conditions: The kernel checks for the presence of the Apple-only dynamic code-signing entitlement. Even then, only a single mmap call can be made to request an executable and writable page, which is given a randomized address. Safari uses this functionality for its JavaScript JIT compiler.

Supporting extensions

iOS and iPadOS allow apps to provide functionality to other apps by providing extensions. Extensions are special-purpose signed executable binaries packaged within an app. During installation, the system automatically detects extensions and makes them available to other apps using a matching system.

Extension points

A system area that supports extensions is called an extension point. Each extension point provides APIs and enforces policies for that area. The system determines which extensions are available based on extension point–specific matching rules. The system automatically launches extension processes as needed and manages their lifetime. Entitlements can be used to restrict extension availability to particular system apps. For example, a Today view widget appears only in Notification Center, and a sharing extension is available only from the Sharing pane. Examples of extension points are Today widgets, Share, Actions, Photo Editing, File Provider, and Custom Keyboard.

How extensions communicate

Extensions run in their own address space. Communication between the extension and the app from which it was activated uses interprocess communications mediated by the system framework. They don't have access to each other's files or memory spaces. Extensions are designed to be isolated from each other, from their containing apps, and from the apps that use them. They are sandboxed like any other third-party app and have a container separate from the containing app's container. However, they share the same access to privacy controls as the container app. So if a user grants Contacts access to an app, this grant is extended to the extensions that are embedded within the app, but not to the extensions activated by the app.

How custom keyboards are used

Custom keyboards are a special type of extension since they're enabled by the user for the entire system. Once enabled, a keyboard extension is used for any text field except the passcode input and any secure text view. To restrict the transfer of user data, custom keyboards run by default in a very restrictive sandbox that blocks access to the network, to services that perform network operations on behalf of a process, and to APIs that would allow the extension to exfiltrate typing data. Developers of custom keyboards can request that their extension have Open Access, which lets the system run the extension in the default sandbox after getting consent from the user.

MDM and extensions

For devices enrolled in a mobile device management (MDM) solution, document and keyboard extensions obey Managed Open In rules. For example, the MDM solution can prevent a user from exporting a document from a managed app to an unmanaged Document Provider, or using an unmanaged keyboard with a managed app. Additionally, app developers can prevent the use of third-party keyboard extensions within their app.

Adopting Data Protection in apps

The iOS Software Development Kit (SDK) for iOS and iPadOS offers a full suite of APIs that make it easy for third-party and in-house developers to adopt Data Protection and help ensure the highest level of protection in their apps. Data Protection is available for file and database APIs, including NSFileManager, CoreData, NSData, and SQLite.

The Mail app database (including attachments), managed books, Safari bookmarks, app launch images, and location data are also stored through encryption, with keys protected by the user's passcode on their device. Calendar (excluding attachments), Contacts, Reminders, Notes, Messages, and Photos implement the Data Protection entitlement Protected Until First User Authentication.

User-installed apps that don't opt in to a specific Data Protection class receive Protected Until First User Authentication by default.

Joining an App Group

Apps and extensions owned by a given developer account can share content when configured to be part of an App Group. It's up to the developer to create the appropriate groups on the Apple Developer Portal and include the desired set of apps and extensions. Once configured to be part of an App Group, apps have access to the following:

- A shared on-volume container for storage, which stays on the device as long as at least one app from the group is installed
- Shared preferences
- Shared Keychain items

The Apple Developer Portal ensures that App Group IDs (GID) are unique across the app ecosystem.

Verifying accessories

The Made for iPhone, iPad, and iPod touch (MFi) licensing program provides vetted accessory manufacturers access to the iPod Accessories Protocol (iAP) and the necessary supporting hardware components.

When an MFi accessory communicates with an iOS or iPadOS device using a Lightning connector or through Bluetooth, the device asks the accessory to prove it has been authorized by Apple by responding with an Apple-provided certificate, which is verified by the device. The device then sends a challenge, which the accessory must answer with a signed response. This process is entirely handled by a custom integrated circuit (IC) that Apple provides to approved accessory manufacturers and is transparent to the accessory itself.

Accessories can request access to different transport methods and functionality—for example, access to digital audio streams over the Lightning cable, or location information provided over Bluetooth. An authentication IC ensures that only approved accessories are granted full access to the device. If an accessory doesn’t support authentication, its access is limited to analog audio and a small subset of serial (UART) audio playback controls.

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 AES-128 in 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

macOS app security overview

App security on 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. It offers technologies to detect and remove malware, and offers additional protections designed to prevent untrusted apps from accessing user data. Ultimately, macOS users are free to operate within the security model that makes sense for them—even 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.

On 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

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 has not been altered. Gatekeeper also requests user approval before opening downloaded software for the first time to make sure the user has not 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 as well as the notarization pipeline ensure apps contain no known malware. Therefore, *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. Furthermore, users can override Gatekeeper's 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 protects against the distribution of malicious plug-ins with benign apps, where using the app triggers loading a malicious plugin 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

XProtect

macOS includes built-in state-of-the-art antivirus technology called XProtect for the signature-based detection of malware, the use of which supports best-practice protection from viruses and malware. The system uses YARA signatures, which Apple updates regularly. Apple monitors for new malware infections and strains, and updates signatures automatically—independent from system updates—to help defend Mac computers 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
- Has been changed

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.

Malware Removal Tool

Should malware make its way onto a Mac, macOS also includes technology to remediate infections. In addition to monitoring for malware activity in the ecosystem to be able to revoke Developer IDs (if applicable) and issue XProtect updates, Apple also issues updates to macOS to remove malware from any impacted systems that are configured to receive automatic security updates. When the Malware Removal Tool (MRT) receives updated information, malware is removed—sometimes after the next restart. MRT doesn’t automatically reboot the Mac.

Automatic security updates

Apple issues the updates for XProtect and the malware removal tool automatically based on the latest threat intelligence available. By default, macOS checks for these updates daily. For more information on automatic security updates, see the Apple Support article Automatic security updates.

Controlling app access to files

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 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 in iOS, iPadOS, macOS, 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-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 usingTouch 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, asTouch ID and Face ID aren’t available when changing the passphrase. After choosing a new passphrase, the Notes app rewraps the keys of all existing notes in the same account 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 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 has not 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 include Apple ID, iCloud, Sign in with Apple, Apple Pay, iMessage, FaceTime, and Find My.

These services provide powerful capabilities for cloud storage and sync, authentication, payment, messaging, communications, and more, all while protecting users' privacy and the security of their data.

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

Apple ID and Managed Apple ID

Apple ID and Managed Apple ID overview

An Apple ID is the account that is used to sign in to Apple services such as iCloud, iMessage, FaceTime, the iTunes Store, App Store, Apple TV app, Book Store, 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 with Apple ID

To help users further secure their accounts, Apple offers two-factor authentication—an extra layer of security for Apple IDs. It is 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 is 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 to the user's iCloud account, requesting approval of the web session. In cases where the user is signing into an Apple website from a browser on a trusted device, they see the code displayed locally on the device they are using. Entering it approves the web session using the user's trusted device.

Account recovery

An Apple ID account can be restored if the password is forgotten by using a trusted device to reset the Apple ID password. If a trusted device isn't available but the password is known, a trusted phone number can be used to authenticate through SMS verification. In addition, a previously used passcode can be used to reset Apple ID in conjunction with SMS verification to provide immediate recovery for an Apple ID. If these options are not possible, then the account recovery process must be followed. See the Apple Support article Recover your Apple ID when you can't reset your password.

Two-step verification with Apple ID

Since 2013, Apple has also offered a similar security method called two-step verification. When two-step verification is enabled, the user's identity must be verified using a temporary code sent to one of the user's trusted devices. Two-step verification is required before changes are permitted to their Apple ID account information; before signing in to iCloud, iMessage, FaceTime, or Game Center; and before a purchase is made with a new device from the iTunes Store, the App Store, the Apple TV app, or Apple Books. Users are also provided with a 14-character Recovery Key to be stored in a safe place in case they ever forget their password or lose access to their trusted devices. While most new users are encouraged to use two-factor authentication, there are still some situations where two-step verification is recommended instead.

Managed Apple IDs

Managed Apple IDs function much like an Apple ID, but are owned and controlled by enterprise or educational institutions. 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 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 inspecting 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 that serves as the second factor of the Apple ID two-factor authentication process. While 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 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 AES-128 and a key derived from each chunk's contents, with the keys using SHA-256. 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

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 when 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 backed 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.

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.

iCloud Backup contents

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 Photo Library in iOS 8.1, iPadOS 13.1, 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.
    - iOS 8.1 or later
    - iPadOS 13.1
    - OS X 10.10.3 or later
- 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 subsequently 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 regardless of whether the user restores a 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.

Passcode and password management

Passcode and password management overview

iOS, iPadOS, and macOS offer a number of features to make it easy for users to securely and conveniently authenticate to third-party apps and websites that use passwords for authentication. 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. For sites that don't support Sign in with Apple, Automatic Strong Password enable a user's devices to automatically create, sync, and enter unique strong passwords for sites and apps. Passwords are saved to a special Password AutoFill Keychain that is user controlled and manageable, in iOS and iPadOS, by going to Settings > Passwords & Accounts > Website & App 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

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 Apple's new 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. 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 anti-fraud signal into Sign in with Apple that 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 Passwords is also available in apps. Users must opt out of using strong passwords. Generated passwords are saved in the keychain and synchronized 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 password context as for 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

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

- Filling 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 how Password AutoFill works in Safari. In iOS and iPadOS, users do this by tapping 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 using the same app-website association mechanism, powered by the 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, but without 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 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 passcodes

iOS and iPadOS apps can interact with the Password AutoFill Keychain using the following two APIs:

- SecRequestSharedWebCredential
- SecAddSharedWebCredential

iOS, iPadOS, and macOS apps can request the Password AutoFill Keychain's help with signing a user in using ASAuthorizationPasswordProvider. The Password provider and its request can be used in conjunction with Sign in with Apple, so that the same API call 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 reuse and strength auditing

The Password AutoFill Keychain passwords list in iOS, iPadOS, and macOS indicates which of a user's saved passwords will be reused with other websites, as well as passwords that are considered weak.

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 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 brute force a password.

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.”

Weak and reused passwords are indicated in the list of passwords. If the user logs into a website in Safari using a previously saved password that is very weak, such as one of the most common passwords, they are shown an alert strongly encouraging them to upgrade to an Automatic Strong Password.

Sending passwords to other users or devices

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.

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

iCloud Keychain

iCloud Keychain overview

iCloud Keychain 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.

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.

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 is used to sign the application ticket. Finally, the application ticket is placed in iCloud.

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 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.

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.

iCloud Keychain recovery

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.

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.

Many users next want to escrow their keychain with Apple. The process is that the iOS, iPadOS, or macOS device exports a copy of the user’s Keychain, 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.

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.

In addition to 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 that must be replied to in order 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 HSM clusters 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 key used to encrypt the user’s Keychain. With that key, the Keychain—retrieved from iCloud key value storage—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.

Safari integration with iCloud Keychain

Safari can automatically generate cryptographically strong random strings for website passwords, which are stored in 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.

Apple Pay

Apple Pay overview

With Apple Pay, users can use supported iOS devices, iPad, Mac, and Apple Watch 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 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 components

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 a unique IC and platform certificate.

Beside the payment industry evaluation requirement, Apple has also certified the Secure Element based on the Common Criteria (CC) standard with the targets of level 6 for the hardware and 5+ for the SE software.

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.

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

Credit, debit, and prepaid card provisioning overview with Apple Pay

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 are encrypted using TLS v1.2.

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, watchOS, and 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.

Add credit or debit cards manually to Apple Pay

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), as well as 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.

Add credit or debit cards from an iTunes Store account to Apple Pay

For a credit or debit card on file with iTunes, the user may be required to reenter theirApple 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.

Add 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.

Additional verification with Apple Pay

A card issuer can decide whether a credit or debit card requires additional verification. Depending on what is 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, the Secure Element allows a payment to be made only after it receives authorization from the Secure Enclave. On iPhone or iPad, this involves confirming 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 is not configured or not 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.

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 is 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's 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.

Transaction authorization

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 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's encryption and anti-rollback mechanism. It's securely delivered to the Secure Element through the pairing key. On receipt of a new AR value, the Secure Element marks any previously added cards as deleted.

Transaction-specific dynamic security code in Apple Pay

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 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.

Pay with credit and debit cards in stores with Apple Pay

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:

- 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.

Pay with credit and debit cards within apps using Apple Pay

Apple Pay can also be used to make payments within iOS, iPadOS, 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 if the device supports Apple Pay and if 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 is 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.

Pay with credit and debit cards on the web using Apple Pay

Apple Pay can be used to make payments on websites with iPhone, iPad, and Apple Watch. 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 all participating websites to register with Apple. The Apple servers perform domain name validation and issue 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 in iOS, iPadOS, and macOS.

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 Wallet Value Added Services protocol (Apple VAS). 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.

In iOS 12 to 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 to be used without authentication.

Render 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 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 OS 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.

Additionally, when a user erases the entire device—using Erase All Content and Settings, using Find My, or restoring their device in —iOS, iPadOS, and macOS devices 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

In iOS 11.2 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.

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.

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.

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.

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 credit or debit card for adding money to Apple Cash, sending money to another user, or providing supplemental money if the Apple Cash balance is insufficient, then an encrypted payment credential is also produced and sent to Apple Pay servers, similar to what is used for Apple Pay 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 has lived on previously)—is securely transmitted to Apple's partner and encrypted using their key. Apple can't decrypt this data.

Apple Card

Apple Card application in the Wallet app

In iOS 12.4 or later, macOS 10.14.6 or later, 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 applying for Apple Card, user identity information is securely verified by Apple's identity provider partners 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 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 physical card is received by the user, the card is activated using the NFC tag present in the bi-fold 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 on 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.

Transit cards in the Wallet app

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 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. The security of reloading the balance when using Apple Pay is described in Pay with credit and debit cards within apps. The process of provisioning the transit card from within the transit card issuer's app is described in Add 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 like 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
- 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 for transit in the Wallet app

In some cities, transit readers accept EMV cards to pay for transit rides, when users present a credit or debit card to those readers, user authentication is required just as “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 > 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 the Wallet app

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 Add 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
- 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 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 thumb's 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 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 is not 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.

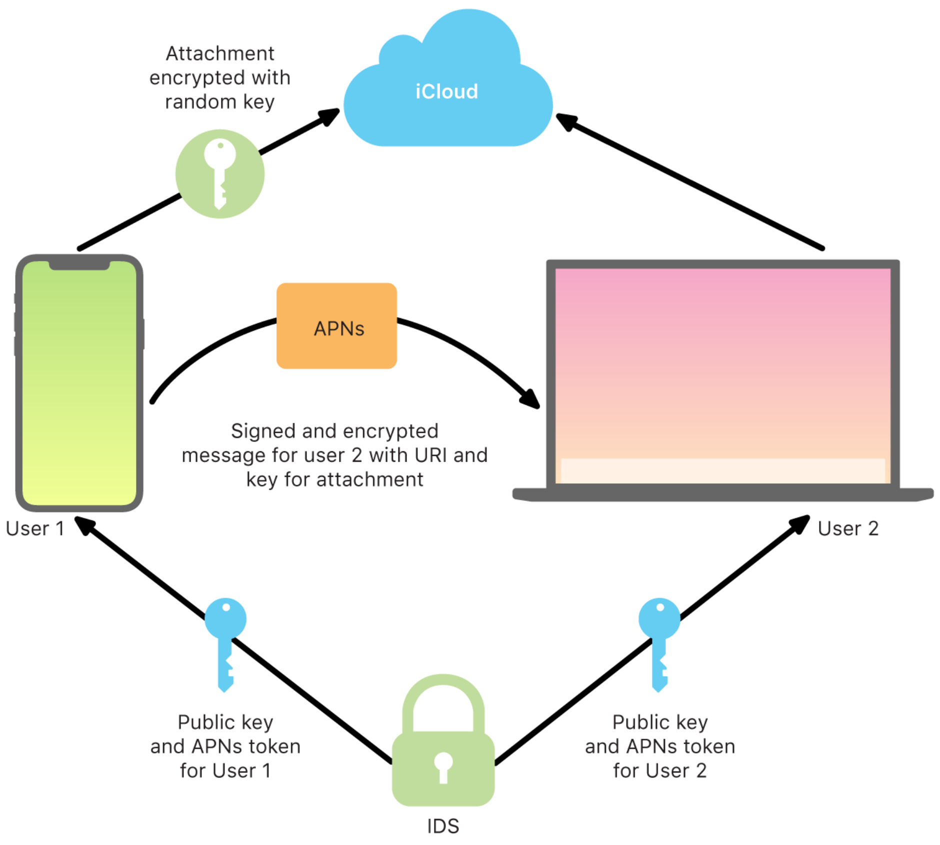
How iMessage sends and receives messages

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, 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 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 ECDSA using the sending device's private signing key. Starting with iOS 13 and iPadOS 13.1, devices may use an 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 4KB or 16KB in size, depending on 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 a 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.


Attachment encrypted with random key
iCloud
APNs
Signed and encrypted message for user 2 with URI and key for attachment
User 1
User 2
Public key and APNs token for User 1
Public key and APNs token for User 2
IDS

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.

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 IV is generated and the data is encrypted using AES-CTR and Key1. 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-bit 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 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 in the iMessage.

Once 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.

Business Chat

Business Chat is a messaging service that enables users to communicate with businesses using the Messages app. Only users can initiate the conversation, and the business receives an opaque identifier for the user. The business doesn’t receive the user’s phone number, email address, or iCloud account information. When the user chats with Apple, Apple receives a Business Chat ID associated with the user’s Apple ID. Users remain in control of whether they want to communicate. Deleting a Business Chat conversation removes it from the user’s Messages app and blocks the business from sending further messages to the user.

Messages sent to the business are individually encrypted between the user’s device and Apple’s messaging servers. Apple’s messaging servers decrypt these messages and relay them to the business over TLS. Businesses’ replies are similarly sent over TLS to Apple’s messaging servers, which then reencrypt the message to the user’s device. As with iMessage, messages are queued for delivery to offline devices for up to 30 days.

FaceTime

FaceTime is Apple’s video and audio calling service. Like iMessage, FaceTime calls also 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 AES-256 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 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 in Find My

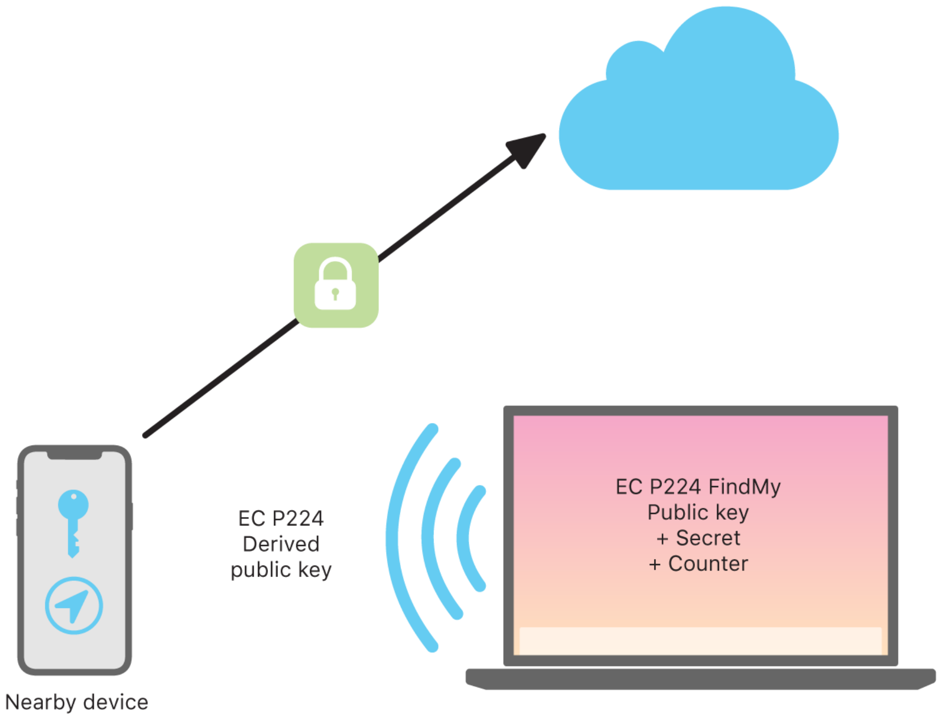
Find My is built on a foundation of advanced public key cryptography. When offline finding is enabled in Find My settings, an EC P-224 private encryption key pair noted {d,P} is generated directly on the device where *d* is the private key and *P* is the public key. Additionally, a 256-bit secret SK_0 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 SKi with the following recursive construction: SK_i= KDF(SK_{i-1}, “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, “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 $P_i = u_i*P + v_i*G$.

When a device goes missing and can’t connect to Wi-Fi or cellular—for example, a MacBook 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.

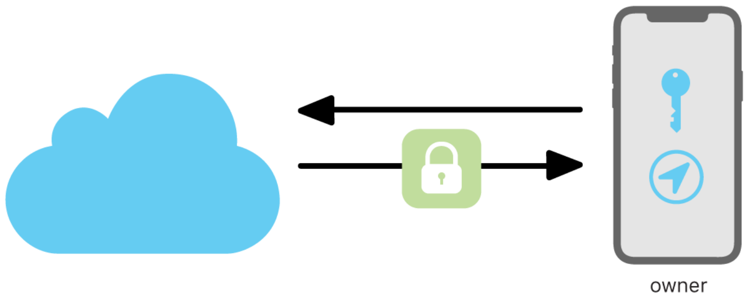
Locating missing devices in Find My

Any Apple devices within Bluetooth range that have offline finding enabled can detect this signal 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 SHA-256 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, SHA-256(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 SHA-256(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 Find My.

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, iPadOS 13.1, and macOS 10.15. This ensures 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.

Keeping users and devices anonymous in Find My

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.

Viewing offline devices in Find My

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. While 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 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

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 AES-256 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 passcodes. 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 (in a similar fashion to 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 completes.

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 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

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 UI. Upon answering 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

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. 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

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, syncing is done using Ed25519 public keys using Apple Identity Service (IDS) between devices in the same iCloud family and then uses ECDH (Curve25519) for the ephemeral key exchange and authenticates it with the Ed25519 public keys. The underlying cryptography is the same as HomeKit devices, except when using Family Sharing, the public keys are synced using IDS.

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 network security

iOS, iPadOS, and macOS support Transport Layer Security (TLS v1.0, TLS v1.1, TLS v1.2, TLS v1.3) and Datagram Transport Layer Security (DTLS). The TLS protocol supports both AES-128 and AES-256, 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 (Network.framework) provide fine-grained control. CFNetwork disallows SSLv3, and apps that use WebKit (such as Safari) are prohibited from making an SSLv3 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 modern, 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 GCM or 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 v1.2 and forward secrecy, and certificates must be valid and signed using SHA-256 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 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 is not available, the trust evaluation will fail.

Virtual Private Networks (VPNs)

Secure network services like virtual private networking typically require minimal setup and configuration to work with iOS, iPadOS, and macOS devices. These devices work with VPN servers that support the following protocols and authentication methods:

- IKEv2/IPSec with authentication by shared secret, RSA Certificates, 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)

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:* Which can be configured for devices managed through an MDM solution and supervised using Apple Configurator 2, Apple School Manager, or Apple Business Manager. This 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 tunneling protocol, 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

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 is supported on the following:

- iPhone 7 or later
- iPad 5th generation or later
- AppleTV 4K or later
- Apple Watch series 3 or later
- Late 2013 and newer Mac computers 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 newer iOS and iPadOS devices.

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:

- iPhone 6 or later
- iPad Air 2 or later
- AppleTV 4th generation or later
- Apple Watch series 3 or later
- Late 2013 and newer Mac computers 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.

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.

All Wi-Fi implementations are **strongly** urged to migrate 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 (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.

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 now 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.

Wi-Fi MAC address randomization support is available on iPhone 5 or later.

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 is initiated while the device is unassociated. This randomization is supported on the following devices:

- iPhone 7 or later
- iPad 5th generation or later
- AppleTV 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 and hidden networks

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 on iOS and iPadOS (with a SIM card) and Internet Sharing on 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.

To mitigate the privacy problems posed by hidden networks, iPhone 6s or later automatically detects when a network is hidden. If the network is not hidden, the iOS or iPadOS device won't send a probe with the SSID included in the request. This prevents the device from broadcasting the name of non-hidden known networks, and hence ensures that it doesn't reveal that it's looking for those networks.

Platform protections

Apple operating systems protect the device from vulnerabilities in network processor firmware, network controllers including Wi-Fi have limited access to application processor memory.

- When USB or SDIO 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 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.

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 (MITM) attacks

Bluetooth version 4.1 added the Secure Connections feature to the 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 is stored in a secure location on iOS, iPadOS, macOS, tvOS, and watchOS devices	Pairing information is stored in a secure location on iOS, iPadOS, macOS, tvOS, and watchOS devices
Authentication	FIPS-approved algorithms (HMAC-SHA-256 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 is used for message integrity	AES-CCM is 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
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 resolving identity key (IRK) exchanged during the pairing procedure.

iOS 13 and iPadOS 13.1 have the ability to derive link keys across transports. 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 Bluetooth Core Specification 4.1 (see Bluetooth Core Specification 5.1).

Ultra Wideband technology

The new Apple-designed U1 chip uses Ultra Wideband technology for spatial awareness—allowing iPhone 11, iPhone 11 Pro, and iPhone 11 Pro Max 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 as other supported Apple devices
- Wi-Fi frame sequence number randomization

Single sign-on

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 (PKINIT) 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 does not 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

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 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 preference 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 ICMP probing and portscan requests

Developer Kits

Developer kits overview

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

- HomeKit
- HealthKit
- CloudKit
- SiriKit
- DriverKit
- ReplayKit
- Camera and ARKit

HomeKit

HomeKit identity

HomeKit provides a home automation infrastructure that uses iCloud and iOS, iPadOS, and macOS security to protect and synchronize 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 synchronized 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 with 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-SHA-512 derived keys. The accessory's MFi certification is also verified during setup. Accessories without an MFi chip can build in support for software authentication on 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-SHA-512 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-SHA-512 derived key, and the data is encrypted using CHACHA20-POLY1305 Authenticated Encryption with Associated Data (AEAD) algorithm. The broadcast encryption key is periodically changed by the iOS, iPadOS, and macOS device and synchronized to other devices using iCloud as described in Data synchronization between devices and users.

HomeKit local data storage

HomeKit stores data about the homes, accessories, scenes, and users on a user's iOS, iPadOS, and macOS device. 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 Data Protection class Protected Until First User Authentication. HomeKit data is only backed up in encrypted backups, so, for example, unencrypted iTunes backups don't contain HomeKit data.

Data synchronization between devices and users

HomeKit data can be synchronized between a user's iOS, iPadOS, and macOS devices using iCloud and iCloud Keychain. The HomeKit data is encrypted during the synchronization using keys derived from the user's HomeKit identity and random nonce. This data is handled as an opaque blob during synchronization. The most recent blob is stored in iCloud to enable synchronization, 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 synchronized 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 synchronized 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, and which room each accessory is in, and other information as detailed in the HomeKit developer documentation at: https://developer.apple.com/homekit/.

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 IP cameras

IP cameras in HomeKit send video and audio streams directly to the iOS, iPadOS, and macOS device on the local network accessing the stream. The streams are encrypted using randomly generated keys on the iOS, iPadOS, and macOS device and the IP camera, which are exchanged over the secure HomeKit session to the camera. When an iOS, iPadOS, or macOS 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 and functions only as a relay between the iOS, iPadOS, and macOS device and the IP camera. When an app displays the HomeKit IP camera video view to the user, HomeKit is rendering the video frames securely from a separate system process so 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-SHA-512 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 AES-256 key. HomeKit also generates poster frames for each clip and these poster frames are encrypted using the same AES-256 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.

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 routers

Routers that support HomeKit allow the user to 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. They also support PPSK authentication, so accessories can be added to the Wi-Fi network using a key that is specific to the accessory and can be revoked when needed. This improves security by not exposing the main Wi-Fi password to accessories, as well as 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-F i network by HomeKit in a Home that has been configured with a HomeKit router. (Accessories that were added to the Wi-Fi prior to adding the router retain their existing credentials.)

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

iCloud remote access for HomeKit accessories

Some legacy HomeKit accessories still require the ability to connect directly with iCloud to enable iOS, iPadOS, and macOS devices to control the accessory remotely when Bluetooth or Wi-Fi communication isn't available. Remote access via a home hub (such as HomePod, Apple TV, or iPad) is preferentially used whenever possible.

iCloud remote access is still supported for legacy devices, and has been carefully designed so that accessories can be controlled and send notifications without revealing to Apple what the accessories are, or what commands and notifications are being sent. HomeKit doesn't send information about the home over iCloud remote access.

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.

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.

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.

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 v1.2 with AES-128-GCM and SHA-256. 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.

HomeKit TV Remote accessories

Third-party HomeKit TV Remote accessories provide Human Interface Design (HID) events and Siri audio to an associated Apple TV added using the Home app. The HID events are sent over the secure session between Apple TV and the Remote. 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 audio frames are sent directly to the Apple TV using a dedicated local network connection between Apple TV and the Remote. The local network connection is encrypted with a per-session HKDF-SHA-512 derived key-pair that is negotiated over the HomeKit session between Apple TV and TV Remote. 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 add their profile to the owner of the home’sApple 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 from 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 theApple 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 synchronize data between users of the home.

HealthKit

HealthKit overview

HealthKit stores and aggregates data from health and fitness apps and healthcare institutions. HealthKit also works directly with health and fitness devices, such as compatible Bluetooth Low Energy (BLE) heart rate monitors and the motion coprocessor built into many iOS devices. All HealthKit interaction with health and fitness apps, healthcare institutions, and health and fitness devices require permission of the user. This data is stored in Data Protection class Protected Unless Open. Access to the data is relinquished 10 minutes after the device locks, and data becomes accessible the next time user enters their passcode or uses Touch ID or Face ID to unlock the device.

HealthKit also aggregates management data, such as access permissions for apps, names of devices connected to HealthKit, and scheduling information used to launch apps when new data is available. This data is stored in Data Protection class Protected Until First User Authentication. Temporary journal files store health records that are generated when the device is locked, such as when the user is exercising. These are stored in Data Protection class Protected Unless Open. When the device is unlocked, the temporary journal files are imported into the primary health databases, then deleted when the merge is completed.

Health data can be stored in iCloud. End-to-end encryption for Health data requiresiOS 12 or later and two-factor authentication. Otherwise, the user’s data is still encrypted in storage and transmission but isn’t encrypted end-to-end. After the user turns on two-factor authentication and update to iOS 12 or later, the user’s Health data is migrated to end-to-end encryption.

If the user backs up their device using iTunes (inmacOS 10.14 or earlier) or the Finder (macOS 10.15 or later), Health data is stored only if the backup is encrypted.

Clinical health records and Health data integrity

Clinical health records

Users can sign into supported health systems within the Health app to obtain a copy of their clinical health records. When connecting a user to a health system, the user authenticates using OAuth 2 client credentials. After connecting, clinical health record data is downloaded directly from the health institution using a TLS v1.3 protected connection. Once downloaded, clinical health records are securely stored alongside other Health data.

Health data integrity

Data stored in the database includes metadata to track the provenance of each data record. This metadata includes an app identifier that identifies which app stored the record. Additionally, an optional metadata item can contain a digitally signed copy of the record. This is intended to provide data integrity for records generated by a trusted device. The format used for the digital signature is the Cryptographic Message Syntax (CMS) specified in RFC 5652.

Health data access by third-party apps

Access to the HealthKit API is controlled with entitlements, and apps must conform to restrictions about how the data is used. For example, apps aren't allowed to use health data for advertising. Apps are also required to provide users with a privacy policy that details its use of health data.

Access to health data by apps is controlled by the user's Privacy settings. Users are asked to grant access when apps request access to health data, similar to Contacts, Photos, and other iOS data sources. However, with health data, apps are granted separate access for reading and writing data, as well as separate access for each type of health data. Users can view, and revoke, permissions they've granted for accessing health data under Settings > Health > Data Access & Devices.

If granted permission to write data, apps can also read the data they write. If granted the permission to read data, they can read data written by all sources. However, apps can't determine access granted to other apps. In addition, apps can't conclusively tell if they have been granted read access to health data. When an app doesn't have read access, all queries return no data—the same response as an empty database would return. This prevents apps from inferring the user's health status by learning which types of data the user is tracking.

Medical ID for users

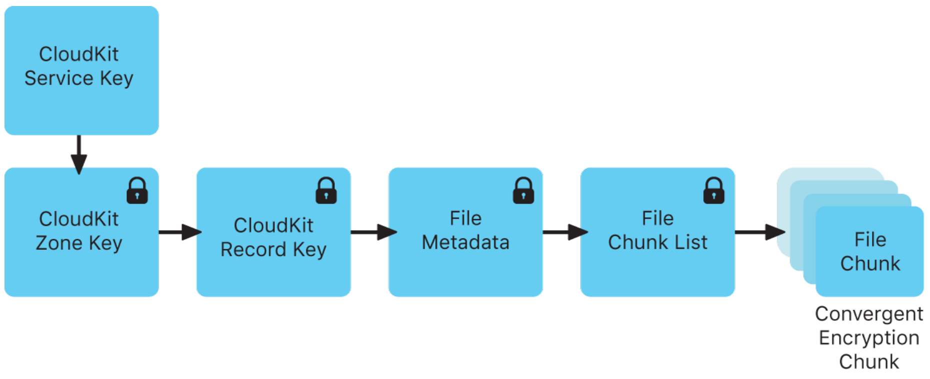
The Health app gives users the option of filling out a Medical ID form with information that could be important during a medical emergency. The information is entered or updated manually and isn't synchronized with the information in the health databases.

The Medical ID information is viewed by tapping the Emergency button on the Lock screen. The information is stored on the device using Data Protection class No Protection so that it's accessible without having to enter the device passcode. Medical ID is an optional feature that enables users to decide how to balance both safety and privacy concerns. This data is backed up in iCloud Backup and isn't synced between devices using CloudKit.

CloudKit

CloudKit allows app developers to 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

Siri uses the app extension system to communicate with third-party apps. Siri on the device 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 does not 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 does not 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

macOS 10.15 uses system extensions to help developers maintain extensions inside their app rather than requiring kernel extensions (“kexts”). This makes for easier installation and increases the stability and security of macOS. 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.

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 IO services, device matching, memory descriptors, and dispatch queues. It also defines IO-appropriate types for numbers, collections, strings, and other common types. The user uses these with family-specific driver frameworks like USBDriverKit and HIDDriverKit.

ReplayKit

ReplayKit movie recording

ReplayKit is a framework that allows 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 UI vended by ReplayKit. The UI 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 recording 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 UI 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 UI that's accessible using Control Center. The UI is implemented using the UIRemoteViewController SPI and is an extension that lives within the ReplayKit framework. It is out-of-process from the hosting app.

- *Upload extension:* The upload 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.

Camera and ARKit

Apple designed cameras with privacy in mind, and third-party apps must obtain the user's consent before accessing 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 the user doesn'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 the user doesn'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.

With iOS 13, iPadOS 13.1, and macOS 10.15, 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 volume. This provides a better balance of security, privacy, and user experience for BYOD programs.

Pairing model

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 synchronization. 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 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 six months, it expires. In iOS 11 or later, this timeframe is shortened to 30 days.

Certain services, including com.apple.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.

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 mobile device management (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 are:

- Allow simple value
- Require alphanumeric value
- Minimum passcode and password length
- Minimum number of complex characters
- Maximum passcode and password age
- Passcode and password history
- Auto-lock timeout
- Grace period for device lock
- Maximum number of failed attempts
- Allow Touch ID or Face ID

Configuration enforcement

A configuration profile is an XML file that allows an administrator to distribute configuration information to iOS, iPadOS, macOS, and tvOS devices. In iOS, iPadOS, and tvOS, most settings that are defined by an installed configuration profile can't be changed by the user. If the user deletes a configuration profile, all the settings defined by the profile are also removed. In this manner, administrators can enforce settings by tying policies to Wi-Fi and data access. For example, a configuration profile that provides an email configuration can also specify a device passcode policy. Users won't be able to access mail unless their passcode meets the administrator's requirements.

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)
- Wi-Fi settings
- VPN 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 3852, supporting 3DES and AES-128.

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 on 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.

Profile removal

Removing configuration profiles depend on how they were installed. The following sequence indicates how a configuration profile can be removed:

1. All profiles can be removed by wiping the device of all data.
2. If the profile is assigned to the device using Apple School Manager or Apple Business Manager, it can be removed by the MDM solution and, optionally, by the user.
3. If the profile is installed by an MDM solution, it can be removed by that specific MDM solution or by the user unenrolling from MDM by removing the enrollment configuration profile.
4. If the profile is installed on a supervised device using Apple Configurator 2, that supervising instance of Apple Configurator 2 can remove the profile.
5. If the profile is installed on a supervised device manually or using Apple Configurator 2 and the profile has a removal password payload, the user must enter the removal password to remove the profile.
6. All other profiles can be removed by the user.

   An account installed by a configuration profile can be removed by removing the profile. A Microsoft Exchange ActiveSync account, including one installed using a configuration profile, can be removed by the Microsoft Exchange Server by issuing the account-only remote wipe command.

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. Since many enterprise users 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.

Mobile device management (MDM)

Apple operating systems support mobile device management (MDM), which allow organizations to securely configure and manage scaled Apple device deployments. MDM capabilities are built on existing OS 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. No confidential or proprietary information is transmitted with APNs.

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, a new enrollment type has been added in iOS 13, iPadOS 13.1, and macOS 10.15—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 limited privileges than unsupervised device enrollments, 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, and the two don't interact with each other.

- *Device Enrollment:* Device Enrollment allows organizations to manually enroll devices and manage many different aspects of device use, including the ability to erase the device. If a user removes the MDM profile, all settings and apps that are being managed by the MDM solution are removed.

- *Automated Device Enrollment:* Automated Device Enrollment lets organizations configure and manage Apple devices from the moment the devices are removed from the box (known as zero-touch deployment). These devices become supervised, and the MDM profile can't be removed by the user. Automated Device Enrollment is designed for devices owned by the organization.

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 a mobile device management (MDM) solution, administrators, employees, staff, and teachers can configure device settings and buy and distribute apps and books. Apple School Manager integrates with Student Information Systems (SISs), SFTP, and Microsoft Azure AD using federated authentication, so administrators can quickly create accounts with school rosters and classes.

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 has received certifications of conformance for Apple School Manager and Apple Business Manager.

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.

Apple Configurator 2

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 before handing them out 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.

Device supervision

During device setup, an organization can configure that device to be supervised. Supervision denotes that the device is owned by the organization, which provides additional control over its configuration and restrictions. With Apple School Manager or Apple Business Manager, supervision can be wirelessly enabled on the device as part of the mobile device management (MDM) enrollment process for iOS, iPadOS, macOS, and tvOS devices, or enabled manually using Apple Configurator 2 for iOS, iPadOS, and tvOS devices. In iOS, iPadOS, and tvOS, supervising a device requires the device to be erased.

The following devices can be supervised:

- iPhone, iPad, and iPod touch with iOS 5 or later
- Apple TV with tvOS 10.2 or later

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

- iOS devices with iOS 13 or later
- iPad with iPadOS 13.1 or later
- Apple TV with tvOS 13 or later
- Mac computers with macOS 10.15 or later

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 the device. Restrictions are sent to devices in a restrictions payload, which is part of a configuration profile. Restrictions can be applied to iOS, iPadOS, macOS, and tvOS devices. Certain restrictions on an iPhone may be mirrored on a paired Apple Watch.

Activation Lock

When Find My is turned on, an iPhone, iPad, iPod touch, Mac or Apple Watch can't be reactivated without entering the owner's Apple ID credentials or the previous passcode or password of the device.

For devices used with a Managed Apple ID created through Apple School Manager or Apple Business Manager, Activation Lock can be tied to an Apple School Manager or Apple Business Manager administrator's Managed Apple ID rather than the user's Apple ID, or disabled using the device's bypass code.

Activation Lock with supervised devices

If the devices are in Apple School Manager or Apple Business Manager and are supervised, Activation Lock can be enabled or disabled using a mobile device management (MDM) solution. This allows Activation Lock without requiring the user to sign in to their iCloud account on the device. This allows organizations to benefit from the theft-deterrent functionality of Activation Lock, while preventing the user from removing the device from Activation Lock.

Activation Lock bypass codes

The MDM solution can store a bypass code when Activation Lock is enabled. This bypass code can be used later to clear Activation Lock automatically when the device needs to be erased and assigned to a new user.

The MDM solution can retrieve a bypass code and permit the user to enable Activation Lock on the device based on the following:

- If Find My is turned *on* when the MDM solution allows Activation Lock, Activation Lock is enabled at that point.
- If Find My is turned *off* when the MDM solution allows Activation Lock, Activation Lock is enabled the next time the user activates Find My.

In iOS 13 and iPadOS 13.1, the bypass codes are only available on two occasions:

- Immediately after a device is enrolled in an MDM solution
- Before the passcode is set on the device, even if the device is already paired with Apple Configurator 2 and supervised

Lost Mode, remote wipe, and remote lock

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 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.

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 is not possible in two situations:

- With User Enrollment
- Using Microsoft Exchange ActiveSync when the account that was installed with User Enrollment

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

Shared iPad overview

Shared iPad is a multi-user mode for use in educational iPad deployments. It allows students to share an iPad without sharing documents and data. Each student gets their own home directory, which is created as an APFS volume protected by the user’s credential. Shared iPad requires the use of a Managed Apple ID that is issued and owned by the school. Shared iPad enables a student to sign in to any organizationally owned device that is configured for use by multiple students. Student data is partitioned into separate home directories, each in their own data protection domains and protected by both UNIX permissions and sandboxing.

Sign 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. Once 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 logged 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, where 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 student signs in without federated authentication, the Managed Apple ID is authenticated with

Apple’s identity servers using the SRP protocol. If authentication is successful, a short-lived access token specific to the device is granted. If the student has used the device before, they already have a local user account, which is unlocked using the same credential.

If the student hasn’t used the device before, a new UNIX user ID, an APFS volume with the user’s home directory, and a local keychain are provisioned. If the device isn’t connected to the Internet (say, because the student is on a field trip), authentication can occur against the local account for a limited number of days. In that situation, only users with previously existing local accounts can sign in. After the time limit has expired, students are required to authenticate online, even if a local account already exists.

After the student’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 student’s settings are restored and their documents and data are synced from iCloud.

While the student 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 student 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.

Sign out of Shared iPad

When a student signs out of Shared iPad, the user keybag for the student is immediately locked and all apps are shut down. To accelerate the case of a new student signing in, iOS defers some ordinary sign-out actions temporarily and presents a Login Window to the new student. If a student signs in during this time (approximately 30 seconds), Shared iPad performs the deferred cleanup as part of signing in to the new student 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.

Screen Time

Screen Time is a feature—in iOS 12 or later, iPadOS, and macOS 10.15 or later, and some features of watchOS—that lets a user understand and control their own app and web usage, or that of their children. While Screen Time is not 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.

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

For a user 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 has two-factor authentication enabled (synchronization is off by default). Screen Time replaces the Restrictions feature found in previous versions of iOS.

In iOS 13, iPadOS 13.1, and macOS 10.15, 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 synchronized devices.

Parents and Screen Time

Parents can also use Screen Time in iOS, iPadOS, and macOS devices to understand and control their children's usage. 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 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. Once they are 18 years old (depending on country or region), 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)
- How many times do users ignore a limit, per user type
- How many 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.

Apple’s Conformance History for Platform Security

ISO/IEC 27001 and ISO/IEC 27018 certifications

Apple has received certifications of conformance to ISO/IEC 27001 for implementing an Information Security Management System ISMS and to ISO/IEC 27018 for the protection of personally identifiable information (PII) in public clouds acting as PII processors. These certifications cover the infrastructure, development, and operations supporting the following products and services: iCloud, iMessage, FaceTime, Siri, Apple Push Notification Service, Apple School Manager, iTunes U, Schoolwork, Apple Business Manager, Apple Business Chat and Managed Apple IDs in accordance with the Statement of Applicability v2.3 dated August 1, 2019. Apple’s conformance with the ISO/IEC standards was certified by the British Standards Institution. The BSI website has certificates of conformance for ISO/IEC 27001 and ISO/IEC 27018. To view these certificates, go to:

- https://www.bsigroup.com/en-GB/our-services/certification/certificate-and-client-directory/search-results/?searchkey=company=apple&licencenumber=IS+649475
- https://www.bsigroup.com/en-GB/our-services/certification/certificate-and-client-directory/search-results/?searchkey=company=Apple&licencenumber=PII%20673269

Cryptographic module validations (FIPS 140-X, ISO/IEC 19790)

The cryptographic modules in Apple’s operating systems have been repeatedly validated by the Cryptographic Module Validation Program (CMVP) as conformant with U.S. Federal Information Processing Standards (FIPS) 140-2 following each major release of the operating systems since 2012. For each major release, Apple submits the modules to the CMVP for validation of the cryptographic modules. These modules provide cryptographic operations for Apple platforms and are available for third party apps.

Apple achieved **Security Level 1** for the software based modules: “CoreCrypto Module on Intel” and the “CoreCrypto Kernel Module on Intel” for macOS, “CoreCrypto Module on ARM” and “CoreCrypto Kernel Module on ARM” for all other Apple operating systems.

In 2019, Apple achieved **FIPS Security Level 2** for the embedded hardware module identified as “Apple Secure Enclave Processor (SEP) Secure Key Store (SKS) Cryptographic Module” enabling government approved use of SEP generated and managed keys. Apple will continue to pursue higher levels for the hardware module with each successive major OS release as appropriate.

FIPS 140-3 was approved by the U.S. Department of Commerce in 2019. The most notable change in this version of the standard is the use of ISO/IEC standards, ISO/IEC 19790:2015 and the associated testing standard ISO/IEC 24759:2017. The CMVP have initiated a transition program and have indicated that starting in 2020, cryptographic modules will be validated using FIPS 140-3 as a basis. Apple cryptographic modules will aim to meet the FIPS 140-3 standard as soon as practicable.

Common Criteria Certifications (ISO/IEC 15408)

Since 2015, Apple has achieved Common Criteria (ISO/IEC 15408) certifications for each major iOS release and has expanded coverage to include the following:

- Mobile Device Certification — OS/HW Platform
  - Mobile Device Fundamental Protection Profile *(Platform Certification)*
  - PP-Module for MDM Agent *(MDM Management of the Platform)*
  - Functional Package for TLS *(All TLS communication from and to the Platform)*
  - PP-Module for VPN Client *(Always-on VPN, IKEv2, IPSEC)*
  - Extended Package for Wireless LAN Clients *(Authenticated and Encrypted Wireless Access)*
- Application Certification
  - Application Software *(Contacts)*
  - Extended Package for Web Browsers *(Safari Browser)*
- Coverage for 2019 includes the release of iPadOS with plans for separate certifications for future iPadOS releases.

Apple also plans to expand coverage to include other operating systems starting with macOS.

Apple is taking an active role within the international Technical Communities (iTC) responsible for developing and updating collaborative Protection Profiles (cPPs) focused on evaluating mobile security technologies. Apple continues to evaluate and pursue certifications against cPPs available today and under development.

Additional Government approvals

Where applicable, Apple has submitted certified platforms and services for use in various Classified Programs. Just a few of the notable ones are the following:

Country or region	Title and URL
Australia	ASD’s Evaluated Products List (EPL) https://www.cyber.gov.au/publications/epl
Germany	BSI’s SecurePIM Government SDS (BSI) https://www.bsi.bund.de/DE/Themen/Sicherheitsberatung/ZugelasseneProdukte/mobile_Kommunikation/mobileKommunikation_node.html
United Kingdom	NCSC’s Commercial Product Assurance (CPA) https://www.ncsc.gov.uk/section/products-services/overview
United States	NSA’s Commercial Solutions for Classified (CSfC) https://www.nsa.gov/Resources/Everyone/csfc/

As additional Apple platforms and services undergo Common Criteria certifications, they too will be submitted for inclusion under similar programs.

Security configuration guides, certifications and program updates for Apple

Apple has collaborated with governments and organizations worldwide to develop guides that give instructions and recommendations for maintaining a more secure environment, also known as device hardening for high-risk environments. These guides provide defined and vetted information about how to configure and use built-in features in Apple operating systems for enhanced protection.

For the latest information on Apple product security certifications, validations, and guidance, see:

- Product security certifications, validations, and guidance for iOS
- Product security certifications, validations, and guidance for macOS
- Product security certifications, validations, and guidance for watchOS
- Product security certifications, validations, and guidance for tvOS
- Product security certifications, validations, and guidance for T2 Firmware
- Product security certifications, validations, and guidance for SEP: Secure Key Store

Glossary

Term	Definition
Address Space Layout Randomization (ASLR)	A technique employed by iOS 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. In iOS 5 or later, the position of all system apps and libraries are also randomized, along with all third-party apps compiled as position-independent executables.
AES	Advanced Encryption Standard.
AES crypto 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.
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 iOS and iPadOS devices.
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 a Mac.
Boot Progress Register (BPR)	A set of SoC hardware flags that software can use to track the boot modes the device has entered, such as DFU mode and Recovery mode. Once 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.

Term	Definition
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 iOS. It can also refer to the APIs that apps use to protect files and Keychain items.
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, the following prompt is presented: "iTunes 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."
DMA	Direct memory access enables hardware subsystems to access main memory.
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.
ECDSA	A digital signature algorithm based on elliptic curve cryptography.
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.
eSPI	Enhanced Serial Peripheral Interface bus for synchronous serial communication.

Term	Definition
Exclusive Chip Identification (ECID)	A 64-bit identifier that’s unique to the processor in each iOS 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.
iBoot	Code that loads XNU, as part of the secure boot chain. Depending on the SoC generation, iBoot may be loaded by LLB or directly by the boot ROM.
Integrated circuit (IC)	Also known as a microchip.
Joint Test Action Group (JTAG)	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 iOS and third-party apps to store and retrieve passwords, keys, and other sensitive credentials.

Term	Definition
Key wrapping	Encrypting one key with another. iOS uses NIST AES key wrapping, in accordance with RFC 3394.
Low-Level Bootloader (LLB)	On Mac computers with a two-stage boot architecture, code that's invoked by the Boot ROM, and 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. On 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). On 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 the SoC that controls the interface between the SoC and its main memory.
Mobile device management (MDM)	A service that lets the user remotely manage enrolled devices. Once 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 256-bit key used to encrypt a file on the file system using AES128-XTS, where the 256-bit is split to provide both the 128-bit tweak key and the 128-bit cipher key. The per-file key is wrapped by a class key and is stored in the file's metadata.
Provisioning profile	A plist signed by Apple that contains a set of entities and entitlements allowing apps to be installed and tested on an iOS 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.

Term	Definition
Recovery mode	Recovery mode is used to restore an iOS device or Apple TV if iTunes (for iOS devices-only) doesn’t recognize the user’s device or says it’s in Recovery mode, the screen is stuck on the Apple logo for several minutes with no progress bar, or the connect to iTunes screen appears.
Ridge flow angle mapping	A mathematical representation of the direction and width of the ridges extracted from a portion of a fingerprint.
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 OS can independently change the value of each software seed bit as long as the corresponding lock bit hasn’t been set. Once 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	Hardware subsystem that manages the storage media (solid-state drive).
System Coprocessor Integrity Protection (SCIP)	System coprocessors are CPUs on the same SoC as the application processor.
System on Chip (SoC)	An integrated circuit (IC) that incorporates multiple components into a single chip. The application processor, Secure Enclave and other coprocessors are components of the SoC.
System Software Authorization	Combines cryptographic keys built into hardware with an online service to ensure 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 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.

Term	Definition
T2 DFU mode	Device Firmware Upgrade mode for the Apple T2 Security Chip.
UEFI firmware	Unified Extensible Firmware Interface, a replacement technology for BIOS to connect firmware to the operating system of a computer.
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 is 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.
XNU	The kernel at the heart of the iOS and macOS operating systems. It's assumed to be trusted, and enforces security measures such as code signing, sandboxing, entitlement checking, and ASLR.

Document Revision History

Date	Summary
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, and 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

Date	Summary
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
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
apple.com

028-00205