Apple Platform Security

Contents

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. For example, Apple-designed silicon and security hardware powers critical security features. And software protections work to keep the operating system and third-party apps protected. Finally, services provide a mechanism for secure and timely software updates, power a protected app ecosystem, and facilitate secure communications and payments. As a result, 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 silicon and hardware that forms the foundation for security on Apple devices, including the Secure Enclave, a dedicated AES cryptographic 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 or process 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 kit security:** Framework "kits" 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, help prevent unauthorized use, and enable remote wipe if a device is lost or stolen

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's possible in security and privacy. This year Apple devices with Apple SoC's across the product lineup from Apple Watch to iPhone and iPad, and now Mac, utilize custom silicon to power not only efficient computation, but also security. Apple silicon forms the foundation for secure boot, Touch ID and Face ID, and Data Protection, as well as system integrity features never before featured on the Mac including Kernel Integrity Protection, Pointer Authentication Codes, and Fast Permission Restrictions. These integrity features help prevent common attack techniques that target memory, manipulate instructions, and use javascript on the web. They combine to help make sure that even if attacker code somehow executes, the damage it can do is dramatically reduced.

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 14.5, iPadOS 14.5, macOS 11.3, tvOS 14.5, and watchOS 7.4.

Hardware security and biometrics

Hardware security overview

For software to be secure, it must rest on hardware that has security built in. That's why Apple devices—running iOS, iPadOS, macOS, watchOS, or tvOS—have security capabilities designed into silicon. These capabilities include a CPU that powers system security features, as well as additional silicon that's dedicated to security functions. Security-focused hardware follows the principle of supporting limited and discretely defined functions in order to minimize attack surface. Such components include a boot ROM, which forms a hardware root of trust for secure boot, dedicated AES engines for efficient and secure encryption and decryption, and a Secure Enclave. The *Secure Enclave* is a system on chip (SoC) that is included on all recent iPhone, iPad, Apple Watch, Apple TV and HomePod devices, and on a Mac with Apple silicon as well as those with the Apple T2 Security Chip. The Secure Enclave itself follows the same principle of design as the SoC does, containing its own discrete boot ROM and AES engine. The Secure Enclave also provides the foundation for the secure generation and storage of the keys necessary for encrypting data at rest, and it protects and evaluates the biometric data for Touch ID and Face ID.

Storage encryption must be fast and efficient. At the same time, it can't expose the data (or *keying material*) it uses to establish cryptographic keying relationships. The AES hardware engine solves this problem by performing fast in-line encryption and decryption *as files are written or read*. A special channel from the Secure Enclave provides necessary keying material to the AES engine without exposing this information to the Application Processor (or CPU) or overall operating system. This helps ensure that the Apple Data Protection and FileVault technologies protect users' files without exposing long-lived encryption keys.

Apple has designed secure boot to protect the lowest levels of software against tampering and to allow only trusted operating system software from Apple to load at startup. Secure boot begins in immutable code called the Boot ROM, which is laid down during Apple SoC fabrication and is known as the *hardware root of trust*. On Mac computers with a T2 chip, trust for macOS secure boot begins with the T2. (Both the T2 chip and the Secure Enclave also execute their own secure boot processes using their own separate boot ROM—this is an exact analogue to how the A-series and M1 chips boot securely.)

The Secure Enclave also processes fingerprint and face data from Touch ID and Face ID sensors in Apple devices. This provides secure authentication while keeping user biometric data private and secure. It also enables users to benefit from the security of longer and more complex passcodes and passwords with, in many situations, the convenience of swift authentication for access or purchases.

Apple SoC security

Apple-designed silicon forms a common architecture across all Apple products and now powers Mac as well as iPhone, iPad, Apple TV, and Apple Watch. For over a decade, Apple's world-class silicon design team has been building and refining Apple systems on chip (SoCs). The result is a scalable architecture designed for all devices that leads the industry in security capabilities. This common foundation for security features is only possible from a company that designs its own silicon to work with its software.

Apple silicon has been designed and fabricated to specifically enable the system security features detailed below.

Feature	A10	A11, S3	A12, S4	A13, S5	A14, S6	M1
Kernel Integrity Protection	✅	✅	✅	✅	✅	✅
Fast Permission Restrictions		✅	✅	✅	✅	✅
System Coprocessor Integrity Protection			✅	✅	✅	✅
Pointer Authentication Codes			✅	✅	✅	✅
Page Protection Layer		✅	✅	✅	✅	See Note below.

Note: Page Protection Layer (PPL) requires that the platform execute *only* signed and trusted code; this is a security model that isn't applicable on macOS.

Apple-designed silicon also specifically enables the Data Protection capabilities detailed below.

Feature	A10	A11, S3	A12, S4	A13, S5	A14, M1, S6
Sealed Key Protection (SKP)	✅	✅	✅	✅	✅
recoveryOS – All Data Protection Classes protected	✅	✅	✅	✅	✅
Alternate boots of DFU, Diagnostics, and Update - Class A, B, and C data protected			✅	✅	✅

Secure Enclave

Overview

The Secure Enclave is a dedicated secure subsystem integrated into Apple systems on chip (SoCs). The Secure Enclave is isolated from the main processor to provide an extra layer of security and is designed to keep sensitive user data secure even when the Application Processor kernel becomes compromised. It follows the same design principles as the SoC does—a boot ROM to establish a hardware root of trust, an AES engine for efficient and secure cryptographic operations, and protected memory. Although the Secure Enclave doesn't include storage, it has a mechanism to store information securely on attached storage separate from the NAND flash storage that's used by the Application Processor and operating system.
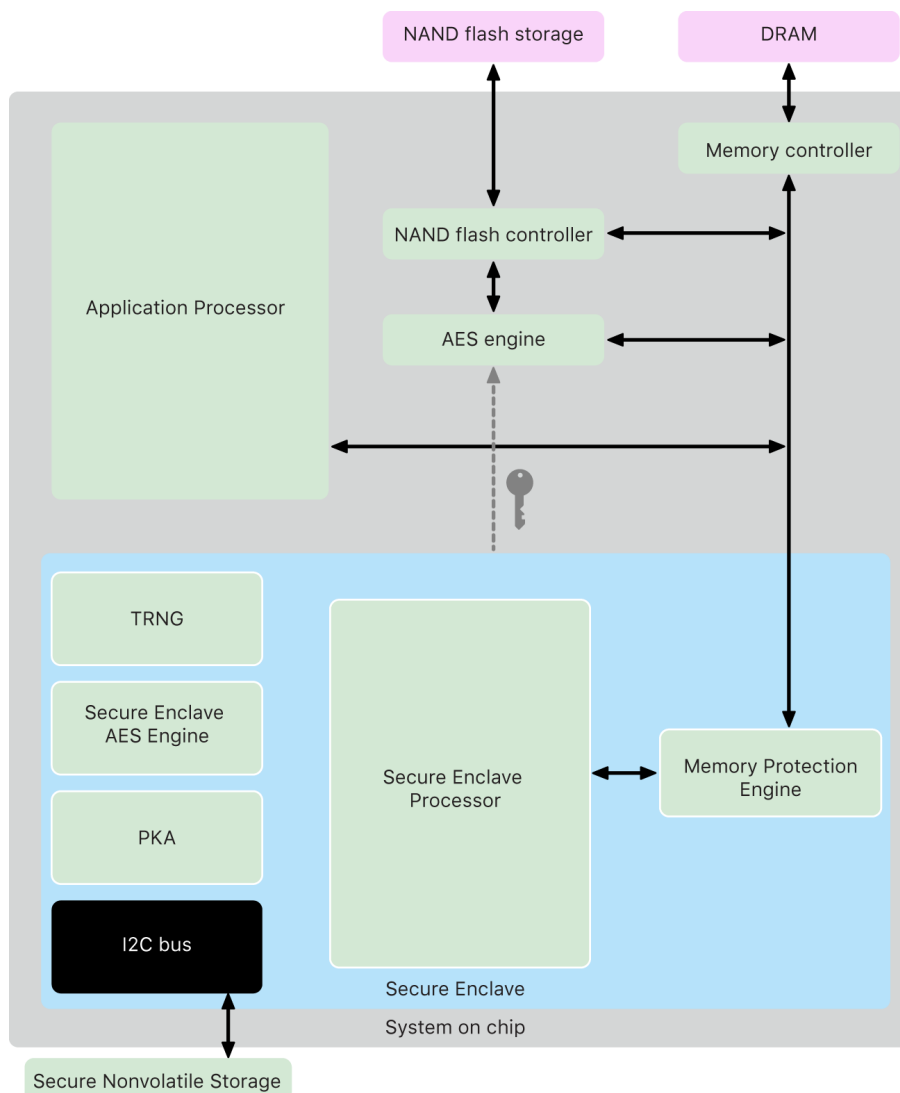
The Secure Enclave components.

The Secure Enclave is a hardware feature of most versions of iPhone, iPad, Mac, Apple TV, Apple Watch, and HomePod—namely:

- iPhone 5s or later

- iPad Air or later

- MacBook Pro computers with Touch Bar (2016 and 2017) that contain the Apple T1 Chip

- Intel-based Mac computers that contain the Apple T2 Security Chip

- Mac computers with Apple silicon

- Apple TV HD or later

- Apple Watch Series 1 or later

- HomePod and HomePod mini

Secure Enclave Processor

The Secure Enclave Processor provides the main computing power for the Secure Enclave. To provide the strongest isolation, the Secure Enclave Processor is dedicated solely for Secure Enclave use. This helps prevent side-channel attacks that depend on malicious software sharing the same execution core as the target software under attack.

The Secure Enclave Processor runs an Apple-customized version of the L4 microkernel. It's designed to operate efficiently at a lower clock speed that helps to protect it against clock and power attacks. The Secure Enclave Processor, starting with the A11 and S4, includes a memory-protected engine and encrypted memory with anti-replay capabilities, secure boot, a dedicated random number generator, and its own AES engine.

Memory Protection Engine

The Secure Enclave operates from a dedicated region of the device's DRAM memory. Multiple layers of protection isolate the Secure Enclave protected memory from the Application Processor.

When the device starts up, the Secure Enclave Boot ROM generates a random ephemeral memory protection key for the Memory Protection Engine. Whenever the Secure Enclave writes to its dedicated memory region, the Memory Protection Engine encrypts the block of memory using AES in Mac XEX (xor-encrypt-xor) mode, and calculates a Cipher-based Message Authentication Code (CMAC) authentication tag for the memory. The Memory Protection Engine stores the authentication tag alongside the encrypted memory. When the Secure Enclave reads the memory, the Memory Protection Engine verifies the authentication tag. If the authentication tag matches, the Memory Protection Engine decrypts the block of memory. If the tag doesn't match, the Memory Protection Engine signals an error to the Secure Enclave. After a memory authentication error, the Secure Enclave stops accepting requests until the system is rebooted.

Starting with the Apple A11 and S4 SoCs, the Memory Protection Engine adds replay protection for Secure Enclave memory. To help prevent replay of security-critical data, the Memory Protection Engine stores a nonce for the block of memory alongside the authentication tag. The nonce is used as an additional tweak for the CMAC authentication tag. The nonces for all memory blocks are protected using an integrity tree rooted in dedicated SRAM within the Secure Enclave. For writes, the Memory Protection Engine *updates* the nonce and each level of the integrity tree up to the SRAM. For reads, the Memory Protection Engine *verifies* the nonce and each level of the integrity tree up to the SRAM. Nonce mismatches are handled similarly to authentication tag mismatches.

On Apple A14, M1, and later SoCS, the Memory Protection Engine supports two ephemeral memory protection keys. The first is used for data private to the Secure Enclave, and the second is used for data shared with the Secure Neural Engine.

The Memory Protection Engine operates inline and transparently to the Secure Enclave. The Secure Enclave reads and writes memory as if it were regular unencrypted DRAM, whereas an observer outside the Secure Enclave sees only the encrypted and authenticated version of the memory. The result is strong memory protection without performance or software complexity tradeoffs.

Secure Enclave Boot ROM

The Secure Enclave includes a dedicated Secure Enclave Boot ROM. Like the Application Processor Boot ROM, the Secure Enclave Boot ROM is immutable code that establishes the hardware root of trust for the Secure Enclave.

On system startup, iBoot assigns a dedicated region of memory to the Secure Enclave. Before using the memory, the Secure Enclave Boot ROM initializes the Memory Protection Engine to provide cryptographic protection of the Secure Enclave protected memory.

The Application Processor then sends the sepOS image to the Secure Enclave Boot ROM. After copying the sepOS image into the Secure Enclave protected memory, the Secure Enclave Boot ROM checks the cryptographic hash and signature of the image to verify that the sepOS is authorized to run on the device. If the sepOS image is properly signed to run on the device, the Secure Enclave Boot ROM transfers control to sepOS. If the signature isn't valid, the Secure Enclave Boot ROM is designed to prevent any further use of the Secure Enclave until the next chip reset.

On Apple A10 and later SoCs, the Secure Enclave Boot ROM locks a hash of the sepOS into a register dedicated to this purpose. The Public Key Accelerator uses this hash for operating-system-bound (OS-bound) keys.

Secure Enclave Boot Monitor

On Apple A13 and later SoCs, the Secure Enclave includes a Boot Monitor designed to ensure stronger integrity on the hash of the booted sepOS.

At system startup, the Secure Enclave Processor's System Coprocessor Integrity Protection (SCIP) configuration helps prevent the Secure Enclave Processor from executing any code other than the Secure Enclave Boot ROM. The Boot Monitor helps prevent the Secure Enclave from modifying the SCIP configuration directly. To make the loaded sepOS executable, the Secure Enclave Boot ROM sends the Boot Monitor a request with the address and size of the loaded sepOS. On receipt of the request, the Boot Monitor resets the Secure Enclave Processor, hashes the loaded sepOS, updates the SCIP settings to allow execution of the loaded sepOS, and starts execution within the newly loaded code. As the system continues booting, this same process is used whenever new code is made executable. Each time, the Boot Monitor updates a running hash of the boot process. The Boot Monitor also includes critical security parameters in the running hash.

When boot completes, the Boot Monitor finalizes the running hash and sends it to the Public Key Accelerator to use for OS-bound keys. This process is designed so that operating system key binding can't be bypassed even with a vulnerability in the Secure Enclave Boot ROM.

True Random Number Generator

The True Random Number Generator (TRNG) is used to generate secure random data. The Secure Enclave uses the TRNG whenever it generates a random cryptographic key, random key seed, or other entropy. The TRNG is based on multiple ring oscillators post processed with CTR_DRBG (an algorithm based on block ciphers in Counter Mode).

Root Cryptographic Keys

The Secure Enclave includes a unique ID (UID) root cryptographic key. The UID is unique to each individual device and isn't related to any other identifier on the device.

A randomly generated UID is fused into the SoC at manufacturing time. Starting with A9 SoCs, the UID is generated by the Secure Enclave TRNG during manufacturing and written to the fuses using a software process that runs entirely in the Secure Enclave. This process protects the UID from being visible outside the device during manufacturing and therefore isn't available for access or storage by Apple or any of its suppliers.

sepOS uses 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. Other protected device-specific secrets include Touch ID or Face ID data. On a Mac, only fully internal storage linked to the AES engine receives 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 in this fashion.

The Secure Enclave also has a device group ID (GID), which is common to all devices that use a given SoC (for example, all devices using the Apple A14 SoC share the same GID).

The UID and GID aren't available through Joint Test Action Group (JTAG) or other debugging interfaces.

Secure Enclave AES Engine

The Secure Enclave AES Engine is a hardware block used to perform symmetric cryptography based on the AES cipher. The AES Engine is designed to resist leaking information by using timing and Static Power Analysis (SPA). Starting with the A9 SoC, the AES Engine also includes Dynamic Power Analysis (DPA) countermeasures.

The AES Engine supports hardware and software keys. Hardware keys are derived from the Secure Enclave UID or GID. These keys stay within the AES Engine and aren't made visible even to sepOS software. Although software can request encryption and decryption operations with hardware keys, it can't extract the keys.

On Apple A10 and newer SoCs, the AES Engine includes lockable seed bits that diversify keys derived from the UID or GID. This allows data access to be conditioned on the device's mode of operation. For example, lockable seed bits are used to deny access to password-protected data when booting from Device Firmware Update (DFU) mode. For more information, see Passcodes and passwords.

AES Engine

Every Apple device with a Secure Enclave also has a dedicated AES256 crypto engine (the "AES Engine") built into the direct memory access (DMA) path between the NAND (nonvolatile) 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's granted access only to memory containing user data through the DMA crypto engine.

At boot time, sepOS generates an ephemeral wrapping key using the TRNG. The Secure Enclave transmits this key to the AES Engine using dedicated wires, designed to prevent it from being accessed by any software outside the Secure Enclave. sepOS can then use the ephemeral wrapping key to wrap file keys for use by the Application Processor file-system driver. When the file-system driver reads or writes a file, it sends the wrapped key to the AES Engine, which unwraps the key. The AES Engine never exposes the unwrapped key to software.

Note: The AES Engine is a separate component from both the Secure Enclave and the Secure Enclave AES Engine, but its operation is closely tied to the Secure Enclave, as shown below.
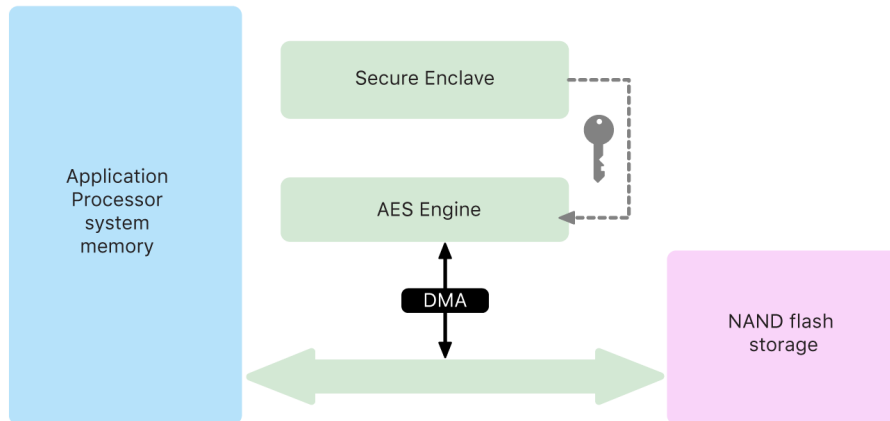
The AES Engine supports line-speed encryption on the DMA path for efficient encryption and decryption of data is it is written and read to storage.

Public Key Accelerator

The Public Key Accelerator (PKA) is a hardware block used to perform asymmetric cryptography operations. The PKA supports RSA and ECC (Elliptic Curve Cryptography) signing and encryption algorithms. The PKA is designed to resist leaking information using timing and side-channel attacks such as SPA and DPA.

The PKA supports software and hardware keys. Hardware keys are derived from the Secure Enclave UID or GID. These keys stay within the PKA and aren't made visible even to sepOS software.

Starting with A13 SoCs, the PKA's encryption implementations have been proved to be mathematically correct using formal verification techniques.

On Apple A10 and later SoCs, the PKA supports OS-bound keys, also referred to as Sealed Key Protection (SKP). These keys are generated using a combination of the device's UID and the hash of the sepOS running on the device. The hash is provided by the Secure Enclave Boot ROM, or by the Secure Enclave Boot Monitor on Apple A13 and later SoCs. These keys are also used to verify the sepOS version when making requests to certain Apple services and are also used to improve the security of passcode-protected data by helping to prevent access to keying material if critical changes are made to the system without user authorization.

Secure nonvolatile storage

The Secure Enclave is equipped with a dedicated secure nonvolatile storage device. The secure nonvolatile storage is connected to the Secure Enclave using a dedicated I2C bus, so that it can only be accessed by the Secure Enclave. All user data encryption keys are rooted in entropy stored in the Secure Enclave nonvolatile storage.

In devices with A12, S4, and later SoCs, the Secure Enclave is paired with a Secure Storage Component for entropy storage. The Secure Storage Component is itself designed with immutable ROM code, a hardware random number generator, a per-device unique cryptographic key, cryptography engines, and physical tamper detection. The Secure Enclave and Secure Storage Component communicate using an encrypted and authenticated protocol that provides exclusive access to the entropy.

Devices first released in Fall 2020 or later are equipped with a 2nd-generation Secure Storage Component. The 2nd-generation Secure Storage Component adds counter lockboxes. Each counter lockbox stores a 128-bit salt, a 128-bit passcode verifier, an 8-bit counter, and an 8-bit maximum attempt value. Access to the counter lockboxes is through an encrypted and authenticated protocol.

Counter lockboxes hold the entropy needed to unlock passcode-protected user data. To access the user data, the paired Secure Enclave must derive the correct passcode entropy value from the user's passcode and the Secure Enclave's UID. The user's passcode can't be learned using unlock attempts sent from a source other than the paired Secure Enclave. If the passcode attempt limit is exceeded (for example, 10 attempts on iPhone), the passcode-protected data is erased completely by the Secure Storage Component.

To create a counter lockbox, the Secure Enclave sends the Secure Storage Component the passcode entropy value and the maximum attempt value. The Secure Storage Component generates the salt value using its random number generator. It then derives a passcode verifier value and a lockbox entropy value from the provided passcode entropy, the Secure Storage Component's unique cryptographic key, and the salt value. The Secure Storage Component initializes the counter lockbox with a count of 0, the provided maximum attempt value, the derived passcode verifier value, and the salt value. The Secure Storage Component then returns the generated lockbox entropy value to the Secure Enclave.

To retrieve the lockbox entropy value from a counter lockbox later, the Secure Enclave sends the Secure Storage Component the passcode entropy. The Secure Storage Component first increments the counter for the lockbox. If the incremented counter exceeds the maximum attempt value, the Secure Storage Component completely erases the counter lockbox. If the maximum attempt count hasn't been reached, the Secure Storage Component attempts to derive the passcode verifier value and lockbox entropy value with the same algorithm used to create the counter lockbox. If the derived passcode verifier value matches the stored passcode verifier value, the Secure Storage Component returns the lockbox entropy value to the Secure Enclave and resets the counter to 0.

The keys used to access password-protected data are rooted in the entropy stored in counter lockboxes. For more information, see Data Protection overview.

The secure nonvolatile storage is used for all anti-replay services in the Secure Enclave. 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

On architectures that don't feature a Secure Storage Component, EEPROM (electrically erasable programmable read-only memory) is utilized to provide secure storage services for the Secure Enclave. Just like the Secure Storage Components, the EEPROM is attached and accessible only from the Secure Enclave, but it doesn't contain dedicated hardware security features nor does it guarantee exclusive access to entropy (aside from its physical attachment characteristics) nor counter lockbox functionality.

Secure Neural Engine

On devices with Face ID, the Secure Neural Engine converts 2D images and depth maps into a mathematical representation of a user's face.

On A11 through A13 SoCs, the Secure Neural Engine is integrated into the Secure Enclave. The Secure Neural Engine uses direct memory access (DMA) for high performance. An input-output memory management unit (IOMMU) under the sepOS kernel's control limits this direct access to authorized memory regions.

Starting with A14 and the M1, the Secure Neural Engine is implemented as a secure mode in the Application Processor's Neural Engine. A dedicated hardware security controller switches between Application Processor and Secure Enclave tasks, resetting Neural Engine state on each transition to keep Face ID data secure. A dedicated engine applies memory encryption, authentication, and access control. At the same time, it uses a separate cryptographic key and memory range to limit the Secure Neural Engine to authorized memory regions.

Power and clock monitors

All electronics are designed to operate within a limited voltage and frequency envelope. When operated outside this envelope, the electronics can malfunction and then security controls may be bypassed. To help ensure that the voltage and frequency stay in a safe range, the Secure Enclave is designed with monitoring circuits. These monitoring circuits are designed to have a much larger operating envelope than the rest of the Secure Enclave. If the monitors detect an illegal operating point, the clocks in the Secure Enclave automatically stop and don't restart until the next SoC reset.

Secure Enclave feature summary

Note: A12, A13, S4, and S5 products first released in Fall 2020 have a 2nd-generation Secure Storage Component; while earlier products based on these SoCs have 1st-generation Secure Storage Component.

SoC	Memory Protection Engine	Secure Storage	AES Engine	PKA
A8	Encryption and authentication	EEPROM	Yes	No
A9	Encryption and authentication	EEPROM	DPA protection	Yes
A10	Encryption and authentication	EEPROM	DPA protection and lockable seed bits	OS-bound keys
A11	Encryption, authentication, and replay prevention	EEPROM	DPA protection and lockable seed bits	OS-bound keys
A12 (Apple devices released before Fall 2020)	Encryption, authentication, and replay prevention	Secure Storage Component gen 1	DPA protection and lockable seed bits	OS-bound keys
A12 (Apple devices released after Fall 2020)	Encryption, authentication, and replay prevention	Secure Storage Component gen 2	DPA protection and lockable seed bits	OS-bound keys
A13 (Apple devices released before Fall 2020)	Encryption, authentication, and replay prevention	Secure Storage Component gen 1	DPA protection and lockable seed bits	OS-bound keys and Boot Monitor
A13 (Apple devices released after Fall 2020)	Encryption, authentication, and replay prevention	Secure Storage Component gen 2	DPA protection and lockable seed bits	OS-bound keys and Boot Monitor
A14	Encryption, authentication, and replay prevention	Secure Storage Component gen 2	DPA protection and lockable seed bits	OS-bound keys and Boot Monitor
S3	Encryption and authentication	EEPROM	DPA protection and lockable seed bits	Yes
S4	Encryption, authentication, and replay prevention	Secure Storage Component gen 1	DPA protection and lockable seed bits	OS-bound keys

SoC	Memory Protection Engine	Secure Storage	AES Engine	PKA
S5 (Apple devices released before Fall 2020)	Encryption, authentication, and replay prevention	Secure Storage Component gen 1	DPA protection and lockable seed bits	OS-bound keys
S5 (Apple devices released after Fall 2020)	Encryption, authentication, and replay prevention	Secure Storage Component gen 2	DPA protection and lockable seed bits	OS-bound keys
S6	Encryption, authentication, and replay prevention	Secure Storage Component gen 2	DPA protection and lockable seed bits	OS-bound keys
T2	Encryption and authentication	EEPROM	DPA protection and lockable seed bits	OS-bound keys
M1	Encryption, authentication, and replay prevention	Secure Storage Component gen 2	DPA protection and lockable seed bits	OS-bound keys and Boot Monitor

Touch ID and Face ID

Touch ID and Face ID security

Passcodes and passwords are essential to the security of Apple devices. At the same time, users require convenient access to their devices, often more than a hundred times a day. Biometric authentication provides a way to retain the security of a strong passcode—or even strengthen the passcode or password because it need not be entered manually— while providing the convenience of swiftly unlocking with a finger press or glance. Touch ID and Face ID don't replace a passcode or password, but in most situations they do make access faster and easier.

Apple's biometric security architecture relies on a strict separation of responsibilities between the biometric sensor and the Secure Enclave, and a secure connection between the two. The sensor captures the biometric image and securely transmits it to the Secure Enclave. During enrollment, the Secure Enclave processes, encrypts, and stores the corresponding Touch ID and Face ID template data. During matching, the Secure Enclave compares incoming data from the biometric sensor against the stored templates to determine whether to unlock the device or respond that a match is valid (for Apple Pay, in-app, and other uses of Touch ID and Face ID). The architecture supports devices that include both the sensor and Secure Enclave (such as iPhone, iPad, and many Mac systems), as well as the ability to physically separate the sensor into a peripheral that is then securely paired to the Secure Enclave in a Mac with Apple silicon.

Touch ID

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 a 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. The channel used to secure this connection varies, depending on whether the Touch ID sensor is built into the device with the Secure Enclave or is located in a separate peripheral.

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

Built-in Touch ID channel security

Communication between the Secure Enclave and the built-in 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's negotiated using a shared key provisioned for each Touch ID sensor and its corresponding Secure Enclave at the factory. For every Touch ID sensor, the shared key is strong, random, and different. The session key exchange uses AES key wrapping, with both sides providing a random key that establishes the session key and uses transport encryption that provides both authentication and confidentiality (using AES-CCM).

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 antispoofing, 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 the user wakes an Apple device that features 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, the Face ID attention check is disabled when VoiceOver is activated and, if required, can be disabled separately.

After the TrueDepth camera confirms the presence of an attentive face, it 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 Secure 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.

Magic Keyboard with Touch ID

The Magic Keyboard with Touch ID (and the Magic Keyboard with Touch ID and Numeric Keypad) provides a Touch ID sensor in an external keyboard that can be used with any Mac with Apple silicon. The Magic Keyboard with Touch ID performs the role of the biometric sensor; it doesn't store biometric templates, perform biometric matching, or enforce security policies (for example, having to enter the password after 48 hours without an unlock). The Touch ID sensor in the Magic Keyboard with Touch ID must be securely paired to the Secure Enclave on the Mac before it can be used, and then the Secure Enclave performs the enrollment and matching operations and enforces security policies in the same way it would for a built-in Touch ID sensor. Apple performs the pairing process in the factory for a Magic Keyboard with Touch ID that is shipped with a Mac. Pairing can also be performed by the user if needed. A Magic Keyboard with Touch ID can be securely paired with only one Mac at a time, but a Mac can maintain secure pairings with up to five different Magic Keyboard with Touch ID keyboards.

The Magic Keyboard with Touch ID and built-in Touch ID sensors are compatible. If a finger that was enrolled on a built-in Mac Touch ID sensor is presented on a Magic Keyboard with Touch ID, the Secure Enclave in the Mac successfully processes the match—and vice versa.

To support secure pairing and thus communication between the Mac Secure Enclave and the Magic Keyboard with Touch ID, the keyboard is equipped with a hardware Public Key Accelerator (PKA) block, to provide attestation, and with hardware-based keys, to perform the necessary cryptographic processes.

Secure pairing

Before a Magic Keyboard with Touch ID can be used for Touch ID operations, it needs to be securely paired to the Mac. To pair, the Secure Enclave on the Mac and the PKA block in the Magic Keyboard with Touch ID exchange public keys, rooted in the trusted Apple CA, and they use hardware-held attestation keys and ephemeral ECDH to securely attest to their identity. On the Mac, this data is protected by the Secure Enclave; on the Magic Keyboard with Touch ID, this data is protected by the PKA block. After secure pairing, all communication between the Mac and the Magic Keyboard with Touch ID is encrypted by AES-GCM, with ephemeral ECDH keys based on the stored identities.

Secure intent to pair

To perform some Touch ID operations for the first time, such as enrolling a new fingerprint, the user must physically confirm their intent to use a Magic Keyboard with Touch ID with the Mac. Physical intent is confirmed by pressing twice on the Mac power button when indicated by the user interface, or by successfully matching a fingerprint that had previously been enrolled with the Mac. For more information, see Secure intent and connections to the Secure Enclave.

Apple Pay transactions can be authorized with a Touch ID match or by entering the macOS user password and pressing twice on the Touch ID button on the Magic Keyboard with Touch ID. The latter enables the user to confirm physical intent even without a Touch ID match.

Magic Keyboard with Touch ID channel security

To help ensure a secure communication channel between the Touch ID sensor in the Magic Keyboard with Touch ID and Secure Enclave on the paired Mac, the following are required:

- The secure pairing between the Magic Keyboard with Touch ID PKA block and the Secure Enclave as described above

- A secure channel between the Magic Keyboard with Touch ID sensor and its PKA block

The secure channel between the Magic Keyboard with Touch ID sensor and its PKA block is established in the factory by using a unique key shared between the two. (This is the same technique used to create the secure channel between the Secure Enclave on the Mac and its built-in sensor, for Mac computers with Touch ID built-in.)

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; instead, they 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 a user's iPhone, iPad, Mac, or Apple Watch cryptographically protects that user's 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 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 pane in System Preferences on Mac

- Unlocking the Users & Groups 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 hasn't yet logged in).

- The user hasn't 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.

- The user exited power off/Emergency SOS by pressing and holding either volume button and the Sleep/Wake button simultaneously for 2 seconds and then pressing Cancel.

- There were 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 button 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 a user is concerned about a false match, Apple recommends using a passcode to authenticate.

Facial matching security

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 infrared (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 types of 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 a 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 or 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.

Uses for Touch ID and Face ID

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:

- *Using Touch ID:* For 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.

- *Using Face ID in stores:* 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.

- *Using Face ID in apps and on the web:* To make a payment within apps and on the web, the user confirms their intent to pay by double-clicking the side button and 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.

Using system-provided APIs

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 with 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 any of the following:

- Require that authentication API operations don't 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 Elliptic Curve Cryptography (ECC) keys inside the 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. When purchases are made, the Secure Enclave verifies that a biometric authorization occurred and then releases ECC keys used to sign the store request.

Secure intent and connections to the Secure Enclave

Secure intent provides a way to confirm a user's intent without any interaction with the operating system or Application Processor. The connection is a physical link—from a physical button to the Secure Enclave—that's available in the following:

- iPhone X or later

- Apple Watch Series 1 or later

- iPad Pro (all models)

- iPad Air (2020)

- Mac computers with Apple silicon

With this link, users can confirm their intent to complete an operation in a way designed such that even software running with root privileges or in the kernel can't spoof.

This feature is used to confirm user intent during Apple Pay transactions and when finalizing pairing Magic Keyboard with Touch ID to a Mac with Apple silicon. A double-press on the appropriate button when prompted by the user interface signals confirmation of user intent. For more information, see Securing purchases with Apple Pay. A similar mechanism—based on the Secure Enclave and T2 firmware—is supported on MacBook models with the Apple T2 Security Chip and no Touch Bar.

Hardware microphone disconnect

All Apple silicon-based Mac notebooks and Intel-based Mac notebooks with the Apple T2 Security Chip feature a hardware disconnect that disables the microphone whenever the lid is closed. On all 13-inch MacBook Pro and MacBook Air notebooks with the T2 chip, all MacBook notebooks with a T2 chip from 2019 or later, and Mac notebooks with Apple silicon, this disconnect is implemented in hardware alone. The disconnect is designed to prevent any software—even with root or kernel privileges in macOS, and even the software on the T2 chip or other firmware—from engaging the microphone when the lid is closed. (The camera isn't disconnected in hardware, because its field of view is completely obstructed with the lid closed.)

iPad models beginning in 2020 also feature the hardware microphone disconnect. When an MFi-compliant case (including those sold by Apple) is attached to the iPad and closed, the microphone is disconnected in hardware. This is designed to prevent microphone audio data being made available to any software—even with root or kernel privileges in iPadOS, or any device firmware.

The protections in this section are implemented directly with hardware logic, according to the following circuit diagram:
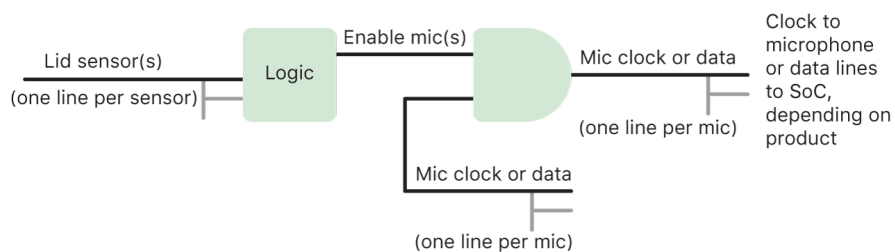
Circuit diagram.

In each product with a hardware microphone cutoff, one or more lid sensors detect the physical closure of the lid or case using some physical property (for example, a Hall effect sensor or a hinge angle sensor) of the interaction. For sensors where calibration is necessary, parameters are set during production of the device and the calibration process includes a nonreversible hardware lock out of any subsequent changes to sensitive parameters on the sensor. These sensors emit a direct hardware signal that goes through a simple set of nonreprogrammable hardware logic. This logic provides debounce, hysteresis, and/or a delay of up to 500 ms before disabling the microphone. Depending on the product, this signal can be implemented either by disabling the lines transporting data between the microphone and the System on Chip (SoC) or by disabling one of the input lines to the microphone module that's allowing it to be active—for example, the clock line or a similar effective control.

Express Cards with power reserve

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 payment or transit card designated as the Express Transit card

- Student ID cards with Express Mode turned on

- Resort passes with Express Mode turned on

- Car keys with Express Mode turned on

Pressing the side button (or on iPhone SE 2nd generation, the Home button) displays the low-battery icon as well as text indicating that 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 only with haptic notification (no visible notification is shown). On iPhone SE 2nd generation, completed transactions may take a few seconds to appear on screen. 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 responsible for controlling access to system resources in Apple devices without compromising usability. System security encompasses the boot-up process, software updates, and protection of computer system resources such as CPU, memory, disk, software programs, and stored data.

The most recent versions of Apple operating systems are the most secure. An important part of Apple security is *secure boot*, which protects the system from malware infection at boot time. Secure boot begins in hardware and builds a chain of trust through software, where each step is designed to ensure 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 timely updates on Apple devices. Subcomponents like the T2 Chip and the Secure Enclave also perform their own secure boot to help ensure they only boot known-good code from Apple. The update system can even help prevent downgrade attacks, so that 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.

Apple devices also include boot and runtime protections so that they maintain their integrity during ongoing operation. Apple-designed silicon on iPhone, iPad, Apple Watch, Apple TV, HomePod, and a Mac with Apple silicon provide a common architecture for protecting operating system integrity. macOS also features an expanded and configurable set of protection capabilities in support of its differing computing model, as well as capabilities supported on all Mac hardware platforms.

Secure boot

Boot process for iOS and iPadOS devices

Each step of the startup process contains components that are cryptographically signed by Apple to enable integrity checking so that boot proceeds only after verifying the chain of trust. These components include the bootloaders, the kernel, kernel extensions, and cellular baseband firmware. This secure boot chain is designed to verify 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 certificate authority (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 checks 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 the Finder (macOS 10.15 or later) or iTunes (in macOS 10.14 or earlier) 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 later SoCs

- *Recovery mode:* Set by iBoot on devices with Apple A10, S2, or later SoCs

On devices with cellular access, a cellular baseband subsystem performs additional secure booting using signed software and keys verified by the baseband processor.

The Secure Enclave also performs a secure boot that checks its software (sepOS) is verified and signed by Apple.

Memory safe iBoot implementation

In iOS 14 and iPadOS 14, Apple modified the C compiler toolchain used to build the iBoot bootloader to improve its security. The modified toolchain implements code designed to prevent memory- and type-safety issues that are typically encountered in C programs. For example, it helps prevent most vulnerabilities in the following classes:

- Buffer overflows, by ensuring that all pointers carry bounds information that is verified when accessing memory

- Heap exploitation, by separating heap data from its metadata and accurately detecting error conditions such as double free errors

- Type confusion, by ensuring that all pointers carry runtime type information that's verified during pointer cast operations

- Type confusion caused by use after free errors, by segregating all dynamic memory allocations by static type

This technology is available on iPhone with Apple A13 Bionic or later, and iPad with the A14 Bionic chip.

Mac computers with Apple silicon

Boot process for a Mac with Apple silicon

When a Mac with Apple silicon is turned on, it performs a boot process much like that of iPhone and iPad.

```
                    ┌─────────────────────────────────────┐
                    │   Boot ROM validates LLB signature   │
                    └─────────────────────────────────────┘
                                      │
                                      ▼
                    ┌─────────────────────────────────────┐
                    │     LLB validates system-paired      │
                    │         firmware signatures          │
                    └─────────────────────────────────────┘
                                      │
┌──────────────────┐                  ▼                  ┌──────────────────┐
│ Secure Enclave   │    ┌─────────────────────────────┐  │ Secure Enclave Boot│
│     signed       │──▶ │ LLB validates LocalPolicy   │◀─│  ROM fetches      │
│   LocalPolicy    │    │        signature            │  │   LocalPolicy     │
│                  │    └─────────────────────────────┘  │ nonces from Secure│
└──────────────────┘                  │                  │ Storage Component │
                                      ▼                  └──────────────────┘
                    ┌─────────────────────────────────────┐
                    │  LLB evaluates iBoot stage 2 signature│
                    │       according to LocalPolicy       │
                    └─────────────────────────────────────┘
                                      │
                                      ▼
                    ┌─────────────────────────────────────┐
                    │ iBoot stage 2 validates macOS-paired │
                    │    firmware, Boot Kernel Collection, │
                    │ Auxilary Kernel Collection (if applicable),│
                    │  system trust cache, and signed system│
                    │     volume signatures, according     │
                    │            to LocalPolicy            │
                    └─────────────────────────────────────┘
                                      │
                                      ▼
                    ┌─────────────────────────────────────┐
                    │               macOS                  │
                    └─────────────────────────────────────┘
```

Boot process steps when a Mac with Apple silicon is started.

The chip executes code from the Boot ROM in the first step in the chain of trust. macOS secure boot on a Mac with Apple silicon verifies not only the operating system code itself, but also the security policies and even kexts (supported, though not recommended) configured by authorized users.

When LLB is launched, it then verifies the signatures and loads system-paired firmware for intra-SoC cores such as the storage, display, system management, and Thunderbolt controllers. LLB is also responsible for loading the LocalPolicy, which is a file signed by the Secure Enclave Processor. The LocalPolicy file describes the configuration that the user has chosen for the system boot and runtime security policies. The LocalPolicy has the same data structure format as all other boot objects, but it's signed locally by a private key that's available only within a particular computer's Secure Enclave instead of being signed by a central Apple server (like software updates).

To help prevent replay of any previous LocalPolicy, LLB must look up a nonce from the Secure Enclave–attached Secure Storage Component. To do this, it uses the Secure Enclave Boot ROM and makes sure the nonce in the LocalPolicy matches the nonce in the Secure Storage Component. This helps prevent an old LocalPolicy—which could have been configured for lower security—from being reapplied to the system after security has been upgraded. The result is that secure boot on a Mac with Apple silicon helps protect not only against rollback of operating system versions but also against security policy downgrades.

The LocalPolicy file captures whether the operating system is configured for Full, Reduced, or Permissive security.

- *Full Security:* The system behaves like iOS and iPadOS, and allows only booting software which was known to be the latest that was available at install time.

- *Reduced Security:* LLB is directed to trust "global" signatures, which are bundled with the operating system. This allows the system to run older versions of macOS. Because older versions of macOS inevitably have unpatched vulnerabilities, this security mode is described as *Reduced*. This is also the policy level required to support booting kernel extensions (kexts).

- *Permissive Security:* The system behaves like Reduced Security in that it uses global signature verification for iBoot and beyond, but it also tells iBoot that it should accept some boot objects being signed by the Secure Enclave with the same key used to sign the LocalPolicy. This policy level supports users that are building, signing, and booting their own custom XNU kernels.

If the LocalPolicy indicates to LLB that the selected operating system is running in Full Security, LLB evaluates the personalized signature for iBoot. If it's running in Reduced Security or Permissive Security, it evaluates the global signature. Any signature verification errors cause the system to boot to recoveryOS to provide repair options.

After LLB hands off to iBoot, it loads macOS-paired firmware such as that for the Secure Neural Engine, the Always On Processor, and other firmware. iBoot also looks at information about the LocalPolicy handed to it from LLB. If the LocalPolicy indicates that there should be an Auxiliary Kernel Collection (AuxKC), iBoot looks for it on the file system, verifies that it was signed by the Secure Enclave with the same key as the LocalPolicy, and verifies that its hash matches a hash stored in the LocalPolicy. If the AuxKC is verified, iBoot places it into memory with the Boot Kernel Collection before locking the full memory region covering the Boot Kernel Collection and AuxKC with the System Coprocessor Integrity Protection (SCIP). If the policy indicates that an AuxKC should be present but it isn't found, the system continues to boot into macOS without it. iBoot is also responsible for verifying the root hash for the signed system volume (SSV), to check that the file system the kernel will mount is fully integrity verified.

Boot modes for a Mac with Apple silicon

A Mac with Apple silicon has the boot modes described below.

Mode	Key combo	Description
macOS	From a shutdown state, press and **release** the power button.	1. Boot ROM hands off to LLB. 2. LLB loads system-paired firmware and the Local Policy for the selected macOS. 3. LLB locks an indication into the Boot Progress Register (BPR) that it's booting into normal macOS, and hands off to iBoot. 4. iBoot loads the macOS-paired firmware, the static trust cache, the device tree, and the Boot Kernel Collection. 5. If the LocalPolicy allows it, iBoot loads the Auxiliary Kernel Collection (AuxKC) of third-party kexts. 6. If the LocalPolicy didn't disable it, iBoot verifies the root signature hash for the signed system volume (SSV).

Mode	Key combo	Description
recoveryOS	From a shutdown state, press and **hold** the power button.	1. Boot ROM hands off to LLB. 2. LLB loads system-paired firmware and the Local Policy for the recoveryOS. 3. LLB locks an indication into the Boot Progress Register that it's booting into recoveryOS, and hands off to iBoot for recoveryOS. 4. iBoot loads the macOS-paired firmware, the trust cache, the device tree, and the Boot Kernel Collection. *Note:* Security downgrades aren't allowed on the recoveryOS LocalPolicy.
Fallback recovery OS	From a shutdown state, **double-press and hold** the power button.	The same process as recoveryOS boot, except that it boots to a second copy of recoveryOS that is kept for resiliency. However, LLB doesn't lock an indication into the Boot Progress Register saying it is going into recoveryOS, and therefore the fallback recovery OS doesn't have the capability to change the system security state.
Safe mode	Boot into recoveryOS per the above, then hold **Shift** while selecting the startup volume.	1. Boots to recoveryOS as per the above. 2. Holding the Shift key while selecting a volume causes the BootPicker application to approve that macOS for booting, as normal, but to also set an nvram variable that tells iBoot to not load the AuxKC on the next boot. 3. System reboots and boots to the targeted volume, but iBoot doesn't load AuxKC.

Startup Disk security policy control for a Mac with Apple silicon

Overview

Unlike security policies on an Intel-based Mac, security policies on a Mac with Apple silicon are for each installed operating system. This means that multiple installed macOS instances with different versions and security policies are supported on the same Mac. For this reason, an operating system picker has been added to Startup Security Utility.
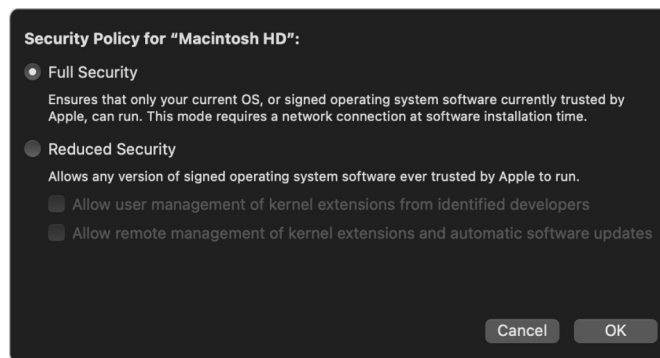

macOS storage selection to change the security policy.

On a Mac with Apple silicon, System Security Utility indicates the overall user-configured security state of macOS, such as the booting of a kext or the configuration of System Integrity Protection (SIP). If changing a security setting would significantly degrade security or make the system easier to compromise, users must enter into recoveryOS by holding the power button (so that malware can't trigger the signal, only a human with physical access can), in order to make the change. Because of this, an Apple-silicon based Mac also won't require (or support) a firmware password—all critical changes are already gated by user authorization. For more information on SIP, see System Integrity Protection.

Full Security and Reduced Security can be set using Startup Security Utility from recoveryOS. But Permissive Security can be accessed only from command-line tools for users who accept the risk of making their Mac much less secure.

Full Security policy

Full Security is the default, and it behaves like iOS and iPadOS. At the time software is downloaded and prepared to install, rather than using the global signature that comes with the software, macOS contacts the same Apple signing server used for iOS and iPadOS and requests a fresh, "personalized" signature. A signature is personalized when it includes the Exclusive Chip Identification (ECID)—a unique ID specific to the Apple CPU in this case—as part of the signing request. The signature given back by the signing server is then unique and usable only by that particular Apple CPU. When the Full Security policy is in effect, the Boot ROM and LLB helps ensure 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.

Security Policy for "Macintosh HD":

◉ 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.

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

☐ Allow user management of kernel extensions from identified developers
☐ Allow remote management of kernel extensions and automatic software updates

[Cancel] [OK]

macOS Full Security policy selection.

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's in security epoch 1 accepts software from security epoch 2, even if the current actual security epoch is 5. With an Apple silicon online signing system, the signing server can reject creating signatures for software that's 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 helps prevent 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 Mac in order to circumvent the protections provided by the latest software. But a user that's in possession of an administrator user name and password for the Mac can always choose the security policy that works best for their use cases.

Reduced Security policy

Reduced Security is similar to Medium Security behavior on an Intel-based Mac with a T2 chip, in which a vendor (in this case, Apple) generates a digital signature for the code to assert it came from the vendor. This design helps prevent attackers from inserting unsigned code. Apple refers to this signature as a "global" signature because it can be used on any Mac, for any amount of time, for a Mac that currently has a Reduced Security policy set. Reduced security doesn't itself provide protection against rollback attacks (although unauthorized operating system changes can result in user data being rendered inaccessible. For more information, see Kernel extensions in a Mac with Apple silicon.
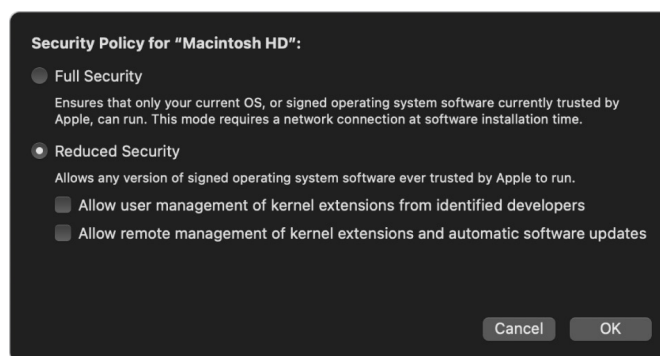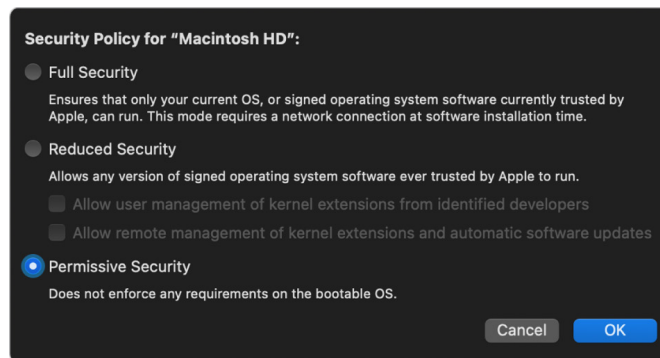

macOS Reduced Security policy selection.

In addition to enabling users to run older versions of macOS, Reduced Security is required for other actions that can put a user's system security at risk, such as introducing third-party kernel extensions (kexts). Kexts have the same privileges as the kernel, and thus any vulnerabilities in third-party kexts can lead to full operating system compromise. This is why developers are being strongly encouraged to adopt system extensions before kext support is removed from macOS for future Mac computers with Apple silicon. Even when third-party kexts are enabled, they can't be loaded into the kernel on demand. Instead, the kexts are merged into an Auxiliary Kernel Collection (AuxKC)—whose hash is stored in the LocalPolicy—and thus they require a reboot. For more information about AuxKC generation, see Kernel extensions in macOS.

Permissive Security policy

Permissive Security is for users who accept the risk of putting their Mac into a much more insecure state. This mode differs from No Security mode on an Intel-based Mac with a T2 chip. With Permissive Security, signature verification is still performed along the entire secure boot chain, but setting the policy to Permissive signals to iBoot that it should accept locally Secure Enclave–signed boot objects, such as a user-generated Boot Kernel Collection built from a custom XNU kernel. In this way, Permissive Security also provides an architectural capability for running an arbitrary "fully untrusted operating system" kernel. When a custom Boot Kernel Collection or fully untrusted operating system is loaded on the system, some decryption keys become unavailable. This is designed to prevent a fully untrusted operating systems from accessing data from trusted operating systems.

Important: Apple doesn't provide or support custom XNU kernels.

macOS Permissive Security policy selection.

There's another way that Permissive Security differs from No Security on an Intel-based Mac with a T2 chip: It's a prerequisite for some security downgrades that in the past have been independently controllable. Most notably, to disable System Integrity Protection (SIP) on a Mac with Apple silicon, a user must acknowledge that they're putting the system into Permissive Security. This is required because disabling SIP has always put the system into a state that makes the kernel much easier to compromise. In particular, disabling SIP on a Mac with Apple silicon disables kext signature enforcement during AuxKC generation time, thus allowing any arbitrary kext to be loaded into kernel memory. Another improvement to SIP that's been made on a Mac with Apple silicon is that the policy store has been moved out of NVRAM and into the LocalPolicy. So now, disabling SIP requires authentication by a user who has access to the LocalPolicy signing key from recoveryOS (reached by pressing and holding the power button). This makes it significantly more difficult for a software-only attacker, or even a physically present attacker, to disable SIP.

It isn't possible to downgrade to Permissive Security from the Startup Security Utility app. Users can downgrade only by running command-line tools from Terminal in recoveryOS, such as `csrutil` (to disable SIP). After the user has downgraded, the fact that it's occurred is reflected in Startup Security Utility, and so a user can easily set the security to a more secure mode.

Note: A Mac with Apple silicon doesn't require or support a specific media boot policy, because technically all boots are performed locally. If a user chooses to boot from external media, that operating system version must first be personalized using an authenticated reboot from recoveryOS. This reboot creates a LocalPolicy file on the internal drive that's used to perform a trusted boot from the operating system stored on the external media. This means the configuration of booting from external media is always explicitly enabled on a per operating system basis, and already requires user authorization, so no additional secure configuration is necessary.

LocalPolicy signing-key creation and management

Creation

When macOS is first installed in the factory, or when a tethered erase-install is performed, the Mac runs code from temporary restore RAM disk to initialize the default state. During this process, the restore environment creates a new pair of public and private keys which are held in the Secure Enclave. The private key is referred to as the *Owner Identity Key (OIK)*. If any OIK already exists, it's destroyed as part of this process. The restore environment also initializes the key used for Activation Lock; the *User Identity Key (UIK)*. Part of that process which is unique to a Mac with Apple silicon is when UIK certification is requested for Activation Lock, a set of requested constraints to be enforced at validation-time on the LocalPolicy are included. If the device can't get a UIK certified for Activation Lock (for example, because the device is currently associated with a Find My Mac account and reported as lost), its unable to proceed further to create a Local Policy. If a device is issued a *User identity Certificate (ucrt)*, that ucrt contains server imposed policy constraints and user requested policy constraints in an X.509 v3 extension.

When an Activation Lock/ucrt is successfully retrieved, it's stored in a database on the server side and also returned to the device. After the device has a ucrt, a certification request for the public key which corresponds to the OIK is sent to the *Basic Attestation Authority (BAA)* server. BAA verifies the OIK certification request using the public key from the ucrt stored in the BAA accessible database. If the BAA can verify the certification, it certifies the public key, returning the *Owner Identity Certificate (OIC)* which is signed by the BAA and contains the constraints stored in ucrt. The OIC is sent back to the Secure Enclave. From then on, whenever the Secure Enclave signs a new LocalPolicy, it attaches the OIC to the Image4. LLB has built-in trust in the BAA root certificate, which causes it to trust the OIC, which causes it to trust the overall LocalPolicy signature.

RemotePolicy constraints

All Image4 files, not just Local Policies, contain constraints on Image4 manifest evaluation. These constraints are encoded using special object identifiers (OIDs) in the leaf certificate. The Image4 verification library looks up the special certificate constraint OID from a certificate during signature evaluation and then mechanically evaluates the constraints specified in it. The constraints are of the form:

- X must exist
- X must not exist
- X must have a specific value

So, for instance, for "personalized" signatures, the certificate constraints will contain "ECID must exist," and for "global" signatures, it will contain "ECID must not exist." These constraints are designed to ensure that all Image4 files signed by a given key must conform to certain requirements to avoid erroneous signed Image4 manifest generation.

In the context of each LocalPolicy, these Image4 certificate constraints are referred to as the *RemotePolicy*. A different RemotePolicy can exist for different boot environments' LocalPolicies. The RemotePolicy is used to restrict the recoveryOS LocalPolicy so that when recoveryOS is booted it can only ever behave as if it's booting with Full Security. This increases trust in the integrity of the recoveryOS boot environment as a place where policy can be changed. The RemotePolicy restricts the LocalPolicy to contain the ECID of the Mac on which the LocalPolicy was generated, and the specific Remote Policy Nonce Hash (rpnh) stored in the Secure Storage Component on that Mac. The rpnh, and therefore the RemotePolicy, change only when actions are taken for Find My Mac and Activation Lock, such as enrollment, unenrollment, remote lock, and remote wipe. Remote Policy constraints are determined and specified at User Identity Key (UIK) certification time and are signed in to the issued User identity Certificate (ucrt). Some Remote Policy constraints, such as ECID, ChipID, and BoardID, are determined by the server. This is designed to prevent one device from signing LocalPolicy files for another device. Other Remote Policy constraints may be specified by the device to help prevent Security downgrade of the Local Policy without providing both the local authentication required to access the current OIK and remote authentication of the account to which the device is Activation Locked.

Contents of a LocalPolicy file for a Mac with Apple silicon

The LocalPolicy is an Image4 file signed by the Secure Enclave. Image4 is an ASN.1 (Abstract Syntax Notation One) DER-encoded data structure format that's used to describe information about secure boot chain objects on Apple platforms. In an Image4-based secure boot model, security policies are requested at software installation time initiated by a signing request to a central Apple signing server. If the policy was acceptable, the signing server returns a signed Image4 file containing a variety of four-character code (4CC) sequences. These signed Image4 files and 4CCs are evaluated at startup by software like the Boot ROM or LLB.

Ownership handoff between operating systems

Access to the Owner Identity Key (OIK) is referred to as "Ownership." Ownership is required to allow users to resign the LocalPolicy after making policy or software changes. The OIK is protected with the same key hierarchy as described in Sealed Key Protection (SKP), with the OIK being protected by the same Key encryption key (KEK) as the Volume encryption key (VEK). This means it's normally protected by both user passwords and measurements of the operating system and policy. There's only a single OIK for all operating systems on the Mac. Therefore, when installing a second operating system, explicit consent is required from the users on the first operating system to hand off Ownership to the users on the second operating system. However, users don't yet exist for the second operating system, when the installer is running from the first operating system. Users in operating systems aren't normally generated until the operating system is booted and the Setup Assistant is running. Thus two new actions are required when installing a second operating system on a Mac with Apple silicon:

- Creating a LocalPolicy for the second operating system
- Preparing an "Install User" for handing off Ownership

When running an Install Assistant and targeting installation for a secondary blank volume, a prompt asks the user if they'd like to copy a user from the current volume to be the first user of the second volume. If the user says yes, the "Install User" which is created is, in reality, a KEK which is derived from the selected user's password and hardware keys, which is then used to encrypt the OIK as it's being handed to the second operating system. Then from within the second operating system Install Assistant, it prompts for that user's password, to allow it to access the OIK in the Secure Enclave for the new operating system. If users opt not to copy a user, the Install User is still created the same way, but an empty password is used instead of a user's password. This second flow exists for certain system administration scenarios. However, users who want to have multi-volume installs and want to perform Ownership handoff in the most secure fashion should always opt to copy a user from the first operating system to the second operating system.

LocalPolicy on a Mac with Apple silicon

For a Mac with Apple silicon, local security policy control has been delegated to an application running in the Secure Enclave. This software can utilize the user's credentials and the boot mode of the primary CPU to determine who can change the security policy and from what boot environment. This helps prevent malicious software from using the security policy controls against the user by downgrading them to gain more privileges.

LocalPolicy manifest properties

The LocalPolicy file contains some architectural 4CCs that are found in most all Image4 files—such as a board or model ID (BORD), indicating a particular Apple chip (CHIP), or Exclusive Chip Identification (ECID). But the 4CCs below focus only on the security policies that users can configure.

Note: Apple uses the term *One True recoveryOS (1TR)* to indicate a boot into the primary recoveryOS which is achieved using a physical power button press. This is different from a normal recoveryOS boot, which can be achieved using NVRAM or which may happen when errors occur on startup. The physical button press increases trust that the boot environment isn't reachable by a software-only attacker who has broken into macOS.

LocalPolicy Nonce Hash (lpnh)

- *Type:* OctetString (48)

- *Mutable environments:* 1TR, recoveryOS, macOS

- *Description:* The lpnh is used for anti-replay of the LocalPolicy. This is an SHA384 hash of the LocalPolicy Nonce (LPN), which is stored in the Secure Storage Component and accessible using the Secure Enclave Boot ROM or Secure Enclave. The raw nonce is never visible to the Application Processor, only to the sepOS. An attacker wanting to convince LLB that a previous LocalPolicy they had captured was valid would need to place a value into the Secure Storage Component, which hashes to the same lpnh value found in the LocalPolicy they want to replay. Normally there is a single LPN valid on the system—except during software updates, when two are simultaneously valid—to allow for the possibility of falling back to booting the old software in the event of an update error. When any LocalPolicy for any operating system is changed, all policies are re-signed with the new *lpnh* value corresponding to the new LPN found in the Secure Storage Component. This change happens when the user changes security settings or creates new operating systems with a new LocalPolicy for each.

Remote Policy Nonce Hash (rpnh)

- *Type:* OctetString (48)

- *Mutable environments:* 1TR, recoveryOS, macOS

- *Description:* The `rpnh` behaves the same way as the `lpnh` but is updated only when the remote policy is updated, such as when changing the state of Find My enrollment. This change happens when the user changes the state of Find My on their Mac.

recoveryOS Nonce Hash (ronh)

- *Type:* OctetString (48)

- *Mutable environments:* 1TR, recoveryOS, macOS

- *Description:* The `ronh` behaves the same way as the `lpnh`, but is found exclusively in the LocalPolicy for recoveryOS. It's updated when the recoveryOS is updated, such as on software updates. A separate nonce from the `lpnh` and `rpnh` is used so that when a device is put into a disabled state by Find My, existing operating systems can be disabled (by removing their LPN and RPN from the Secure Storage Component), while still leaving the recoveryOS bootable. In this way, the operating systems can be reenabled when the system owner proves their control over the system by putting in their iCloud password used for the Find My account. This change happens when a user updates the recoveryOS or creates new operating systems.

Next Stage Image4 Manifest Hash (nsih)

- *Type:* OctetString (48)

- *Mutable environments:* 1TR, recoveryOS, macOS

- *Description:* The *nsih* field represents an SHA384 hash of the Image4 manifest data structure that describes the booted macOS. The macOS Image4 manifest contains measurements for all the boot objects—such as iBoot, the static trust cache, device tree, Boot Kernel Collection, and signed system volume (SSV) volume root hash. When LLB is directed to boot a given macOS, it's designed to ensure that the hash of the macOS Image4 manifest attached to iBoot matches what's captured in the `nsih` field of the LocalPolicy. In this way, the `nsih` captures the user intention of what operating system the user has created a LocalPolicy for. Users change the `nsih` value implicitly when they perform a software update.

Auxiliary Kernel Collection (AuxKC) Policy Hash (auxp)

- *Type:* OctetString (48)

- *Mutable environments:* macOS

- *Description:* The auxp is an SHA384 hash of the user-authorized kext list (UAKL) policy. This is used at AuxKC generation time to help ensure that only user-authorized kexts are included in the AuxKC. smb2 is a prerequisite for setting this field. Users change the auxp value implicitly when they change the UAKL by approving a kext from the Security & Privacy pane in System Preferences.

Auxiliary Kernel Collection (AuxKC) Image4 Manifest Hash (auxi)

- *Type:* OctetString (48)

- *Mutable environments:* macOS

- *Description:* After the system verifies that the UAKL hash matches what's found in the auxp field of the LocalPolicy, it requests that the AuxKC be signed by the Secure Enclave processor application that's responsible for LocalPolicy signing. Next, an SHA384 hash of the AuxKC Image4 manifest signature is placed into the LocalPolicy to avoid the potential for mixing and matching previously signed AuxKCs to an operating system at boot time. If iBoot finds the auxi field in the LocalPolicy, it attempts to load the AuxKC from storage and validate its signature. It also verifies that the hash of the Image4 manifest attached to the AuxKC matches the value found in the auxi field. If the AuxKC fails to load for any reason, the system continues to boot without this boot object and (so) without any third-party kexts loaded. The auxp field is a prerequisite for setting the auxi field in the LocalPolicy. Users change the auxi value implicitly when they change the UAKL by approving a kext from the Security & Privacy pane in System Preferences.

Auxiliary Kernel Collection (AuxKC) Receipt Hash (auxr)

- *Type:* OctetString (48)

- *Mutable environments:* macOS

- *Description:* The auxr is an SHA384 hash of the AuxKC receipt, which indicates the exact set of kexts that were included into the AuxKC. The AuxKC receipt can be a subset of the UAKL, because kexts can be excluded from the AuxKC even if they're user authorized if they're known to be used for attacks. In addition, some kexts that can be used to break the user-kernel boundary may lead to decreased functionality, such as an inability to use Apple Pay or play 4K and HDR content. Users who want these capabilities opt in to a more restrictive AuxKC inclusion. The auxp field is a prerequisite for setting the auxr field in the LocalPolicy. Users change the auxr value implicitly when they build a new AuxKC from the Security & Privacy pane in System Preferences.

CustomOS Image4 Manifest Hash (coih)

- *Type:* OctetString (48)

- *Mutable environments:* 1TR

- *Description:* The coih is an SHA384 hash of CustomOS Image4 manifest. The payload for that manifest is used by iBoot (instead of the XNU kernel) to transfer control. Users change the coih value implicitly when they use the kmutil configure-boot command-line tool in 1TR.

APFS volume group UUID (vuid)

- *Type:* OctetString (16)

- *Mutable environments:* 1TR, recoveryOS, macOS

- *Description:* The vuid indicates the volume group the kernel should use as root. This field is primarily informational and isn't used for security constraints. This vuid is set by the user implicitly when creating a new operating system install.

Key encryption key (KEK) Group UUID (kuid)

- *Type:* OctetString (16)

- *Mutable environments:* 1TR, recoveryOS, macOS

- *Description:* The kuid indicates the volume that was booted. The key encryption key has typically been used for Data Protection. For each LocalPolicy, it's used to protect the LocalPolicy signing key. The kuid is set by the user implicitly when creating a new operating system install.

Paired recoveryOS Trusted Boot Policy Measurement (prot)

- *Type:* OctetString (48)

- *Mutable environments:* 1TR, recoveryOS, macOS

- *Description:* A paired recoveryOS Trusted Boot Policy Measurement (TBPM) is a special iterative SHA384 hash calculation over the Image4 manifest of a LocalPolicy, excluding nonces, in order to give a consistent measurement over time (because nonces like lpnh are frequently updated). The prot field, which is found only in each macOS LocalPolicy, provides a pairing to indicate the recoveryOS LocalPolicy that corresponds to the macOS LocalPolicy.

Has Secure Enclave Signed recoveryOS Local Policy (hrlp)

- *Type:* Boolean

- *Mutable environments:* 1TR, recoveryOS, macOS

- *Description:* The hrlp indicates whether or not the prot value (above) is the measurement of a Secure Enclave–signed recoveryOS LocalPolicy. If not, then the recoveryOS LocalPolicy is signed by the Apple online signing server, which signs things such as macOS Image4 files.

Secure Multi-Boot (smb0)

- *Type:* Boolean

- *Mutable environments:* 1TR, recoveryOS

- *Description:* If smb0 is present and true, LLB allows the next stage Image4 manifest to be globally signed instead of requiring a personalized signature. Users can change this field with Startup Security Utility or bputil to downgrade to Reduced Security.

Secure Multi-Boot (smb1)

- *Type:* Boolean

- *Mutable environments:* 1TR

- *Description:* If smb1 is present and true, iBoot allows objects such as a custom kernel collection to be Secure Enclave signed with the same key as the LocalPolicy. Presence of smb0 is a prerequisite for presence of smb1. Users can change this field using command-line tools such as csrutil or bputil to downgrade to Permissive Security.

Secure Multi-Boot (smb2)

- *Type:* Boolean

- *Mutable environments:* 1TR

- *Description:* If smb2 is present and true, iBoot allows the Auxiliary Kernel Collection to be Secure Enclave signed with the same key as the LocalPolicy. The presence of smb0 is a prerequisite for the presence of smb2. Users can change this field using Startup Security Utility or bputil to downgrade to Reduced Security and enable third-party kexts.

Secure Multi-Boot (smb3)

- *Type:* Boolean

- *Mutable environments:* 1TR

- *Description:* If smb3 is present and true, a user at the device has opted in to mobile device management (MDM) control of their system. Presence of this field makes the LocalPolicy-controlling Secure Enclave processor application accept MDM authentication instead of requiring local user authentication. Users can change this field using Startup Security Utility or `bputil` to enable managed control over third-party kexts and software updates. (In macOS 11.2 or later, MDM can also initiate an update to the latest macOS version if the current security mode is Full Security.)

Secure Multi-Boot (smb4)

- *Type:* Boolean

- *Mutable environments:* recoveryOS, macOS

- *Description:* If smb4 is present and true, the device has opted in to MDM control of the operating system using the Apple School Manager or Apple Business Manager. Presence of this field makes the LocalPolicy-controlling Secure Enclave application accept MDM authentication instead of requiring local user authentication. This field is changed by the MDM solution when it detects that a device's serial number appears in Apple School Manager or Apple Business Manager.

System Integrity Protection (sip0)

- *Type:* 64 bit unsigned integer

- *Mutable environments:* 1TR

- *Description:* The `sip0` holds the existing System Integrity Protection (SIP) policy bits that previously were stored in NVRAM. New SIP policy bits are added here (instead of using LocalPolicy fields like the below) if they're used only in macOS and not used by LLB. Users can change this field using `csrutil` from 1TR to disable SIP and downgrade to Permissive Security.

System Integrity Protection (sip1)

- *Type:* Boolean

- *Mutable environments:* 1TR

- *Description:* If `sip1` is present and true, iBoot will allow failures to verify the SSV volume root hash. Users can change this field using `csrutil` or `bputil` from 1TR.

System Integrity Protection (sip2)

- *Type:* Boolean

- *Mutable environments:* 1TR

- *Description:* If sip2 is present and true, iBoot will not lock the *Configurable Text Read-only Region (CTRR)* hardware register that marks kernel memory as non-writable. Users can change this field using `csrutil` or `bputil` from 1TR.

System Integrity Protection (sip3)

- *Type:* Boolean

- *Mutable environments:* 1TR

- *Description:* If `sip3` is present and true, iBoot will not enforce its built-in allow list for the boot-args NVRAM variable, which would otherwise filter the options passed to the kernel. Users can change this field using `csrutil` or `bputil` from 1TR.

Certificates and RemotePolicy

As described in LocalPolicy signing-key creation and management, the LocalPolicy Image4 also contains the Owner Identity Certificate (OIC) and the embedded RemotePolicy.

Intel-based Mac computers

Boot process for an Intel-based Mac

Intel-based Mac with an Apple T2 Security Chip

When an Intel-based Mac computer with the Apple T2 Security Chip is turned on, the chip performs a secure boot from its Boot ROM in the same fashion as iPhone, iPad, and a Mac with Apple silicon. This verifies the iBoot bootloader and is the first step in the chain of trust. iBoot checks the kernel and kernel extension code on the T2 chip, which then checks the Intel UEFI firmware. The UEFI firmware and the associated signature are initially available only to the T2 chip.
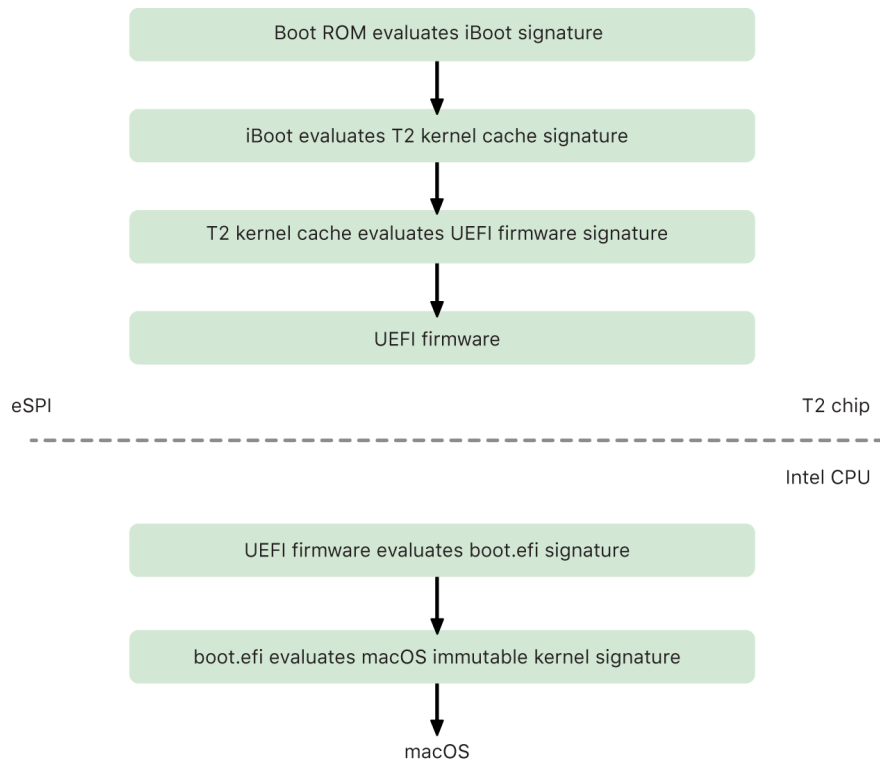
macOS T2 secure boot chain.

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 the 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 Recovery mode, Apple T2 Security Chip Recovery mode, or Apple T2 Security Chip Device Firmware Upgrade (DFU) mode mode.

Microsoft Windows on an Intel-based Mac with a T2 chip

By default, an Intel-based Mac that supports 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 that 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 recoveryOS and informs the user of the trust evaluation failure.

Intel-based Mac computers without a T2 chip

An Intel-based Mac without a T2 chip doesn't support secure boot. Therefore the UEFI firmware loads the macOS booter (boot.efi) from the file system without verification, and the booter loads the kernel (prelinkedkernel) from the file system without verification. To protect the integrity of the boot chain, users should enable all of the following security mechanisms:

- *System Integrity Protection (SIP):* 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 an MDM administrator. This helps prevent 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 helps prevent booting from alternate media, by which an attacker could run code to overwrite the booter.
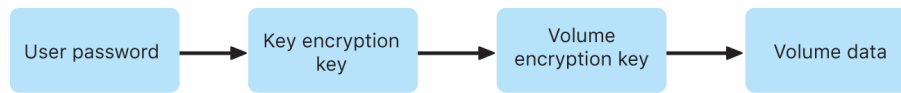
The unlocking process of an Intel-based Mac without a T2 chip.

Boot modes of an Intel-based Mac with an Apple T2 Security Chip

An Intel-based Mac with an Apple T2 Security Chip has 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 presents the Login Window interface, which takes the password to decrypt the storage.
Startup Manager	Option ( )	The UEFI firmware launches the built-in UEFI application that presents the user with a boot device selection interface.
Target Disk Mode (TDM)	T	The UEFI firmware launches the built-in UEFI application that 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 Mac model).
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.

Mode	Key combo	Description
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.
Windows boot	None	If Windows has been installed using Boot Camp, the UEFI firmware hands off to the Windows booter, which hands off to the Windows kernel.

Startup Security Utility on a Mac with an Apple T2 Security Chip

Overview

On an Intel-based Mac with an Apple T2 Security Chip, Startup Security Utility handles a number security policy settings. The utility is accessible by booting into recoveryOS and selecting Startup Security Utility from the Utilities menu and protects supported security settings from easy manipulation by an attacker.
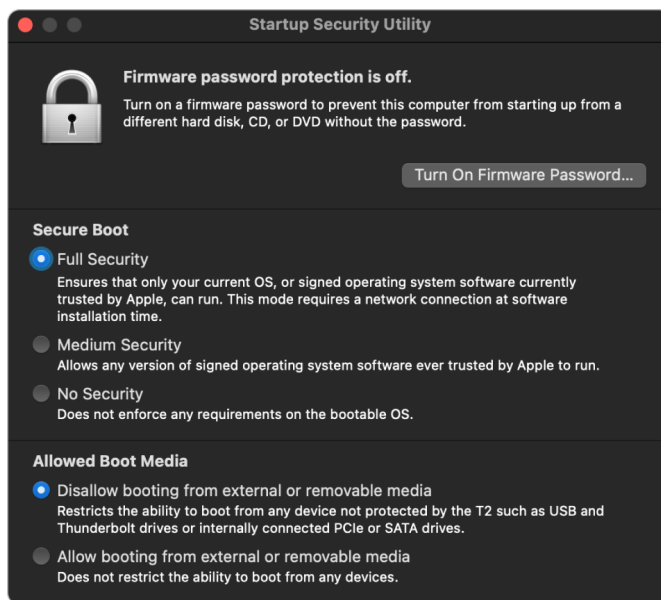
A screenshot of the Startup Security Utility.

Critical policy changes require authentication, even in Recovery mode. 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 recoveryOS. 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 recoveryOS 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. recoveryOS 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 shown only in Startup Security Utility on an Intel-based Mac with a T2 chip. Although most 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 boot policy, and it behaves much like iOS and iPadOS or Full Security on a Mac with Apple silicon. At the time that software is downloaded and prepared to install, it is personalized with a signature that includes the Exclusive Chip Identification (ECID)—a unique ID specific to the T2 chip in this case—as part of the signing request. The signature given back by the signing server is then unique and usable only by that particular T2 chip. The UEFI firmware is designed to ensure that when the Full Security policy is in effect, 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. This helps prevent rollback attacks as described for Full Security on a Mac with Apple silicon.

Medium Security boot policy

Medium Security boot policy is somewhat like a traditional UEFI secure boot, in which 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 a Mac that currently has a Medium Security policy set. Neither iOS, iPadOS, nor the T2 chip itself support global signatures. This setting doesn't attempt to prevent rollback attacks.

Media boot policy

Media boot policy exists only on an Intel-based Mac with a T2 chip and is independent from the secure boot policy. So even if a user disables secure boot, this doesn't change the default behavior of preventing anything other than the storage device directly connected to the T2 chip to boot the Mac. (Media boot policy is not required on a Mac with Apple silicon. For more information, see Startup Disk security policy control.

Firmware password protection in an Intel-based Mac

macOS on Intel-based Mac computers with an Apple T2 Security Chip supports the use of a Firmware Password to help prevent unintended modifications of firmware settings on a specific Mac. The Firmware Password is designed 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.

Note: The firmware password isn't required on a Mac with Apple silicon, because the critical firmware functionality it restricted has been moved into the recoveryOS and (when FileVault is enabled) recoveryOS requires user authentication before its critical functionality can be reached.

The most basic mode of firmware password can be reached from the recoveryOS Firmware Password Utility on an Intel-based Mac *without* a T2 chip, and from the Startup Security Utility on an Intel-based Mac *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.

Setting a Firmware Password is especially important to reduce the risk of attacks on Intel-based Mac computers without a T2 chip from a physically present attacker. The Firmware Password can help prevent an attacker from booting to recoveryOS, from where they could otherwise disable System Integrity Protection (SIP). And by restricting boot of alternative media, an attacker can't execute privileged code from another operating system to attack peripheral firmwares.

A firmware password reset mechanism exists to help users who forget their password. Users press a key combination at startup, and are presented with a model-specific string to provide to AppleCare. AppleCare digitally signs a resource that is signature checked by the Uniform Resource Identifier (URI). If the signature is validated 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 acknowledge that if the password is forgotten and needs removal, the user must bear the cost of the logic board 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 in any of the following ways:

- 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)

recoveryOS and diagnostics environments for an Intel-based Mac

recoveryOS

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 one chunk at a time from the BaseSystem.dmg. This helps ensure that it matches the signed content present in the chunklist. If any of these hashes don't match, booting from the local recoveryOS is aborted and the UEFI firmware attempts to boot from Internet recoveryOS instead.

If the verification is successfully completed, the UEFI firmware mounts the BaseSystem.dmg as a RAM disk and launches the boot.efi file that's in it. There's 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 operating system (of which these elements are only a subset) have already been integrity checked.

Apple Diagnostics

The procedure for booting the local diagnostic environment is mostly the same as launching the recoveryOS. Separate AppleDiagnostics.dmg and AppleDiagnostics.chunklist files are used, but they're verified in the same way as the BaseSystem files are. Instead of launching boot.efi, the UEFI firmware launches a file inside the disk image (.dmg file) 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 diagnostic environment

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. (A user can also specifically request the images to 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 protected against manipulation by an attacker with control of the network. In the event that an individual chunk fails integrity verification, it is re-requested from the OS Recovery Server 11 times, before giving up and displaying an error.

When the internet recovery and diagnostic modes were added to Mac computers in 2011, it was decided that it would be better to use the simpler HTTP transport, and handle content authentication using the chunklist mechanism, rather than implement the more complicated HTTPS functionality in the UEFI firmware, and thus increase the firmware's attack surface.

Secure software updates

Overview

Security is a process; it isn't enough to reliably boot the operating system version installed at the factory—there must also exist a mechanism to quickly and securely obtain the latest security updates. Apple regularly releases software updates to address emerging security concerns. Users of iOS and iPadOS devices receive update notifications on the device. Mac users find available updates in System Preferences. Updates are delivered wirelessly, for rapid adoption of the latest security fixes.

The update process uses the same hardware-based root of trust that secure boot uses that is designed to install only Apple-signed code. The update process also uses system software authorization to check that only copies of operating system versions that are actively being signed by Apple can be installed on iOS and iPadOS devices, or on Mac computers with the Full Security setting configured as the secure boot policy in Startup Security Utility. With these secure processes in place, Apple can stop signing older operating system versions with known vulnerabilities and help prevent downgrade attacks.
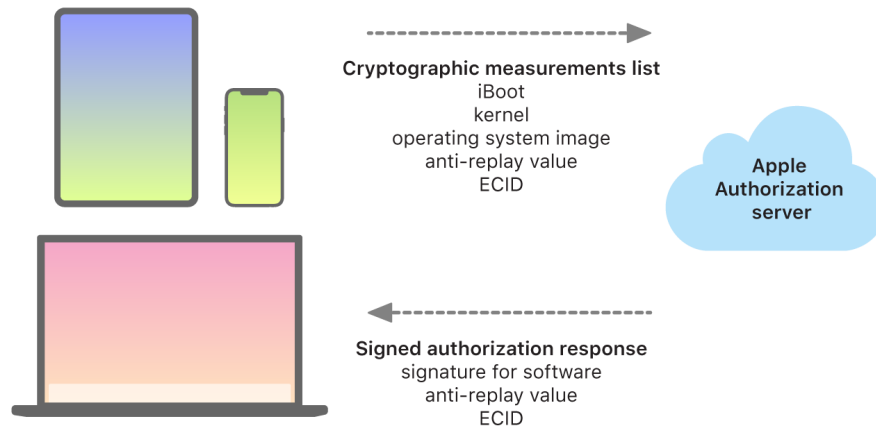
For greater software update security, when the device to be upgraded is physically connected to a Mac, a full copy of iOS or iPadOS is downloaded and installed. But for over-the-air (OTA) software updates, *only the components required to complete an update are downloaded,* improving network efficiency by not downloading the entire operating system. What's more, 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 still need to contact Apple servers to complete the update process.)

Personalized update process

During upgrades and updates, 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 the operating system image), a random anti-replay value (the nonce), and the device's unique Exclusive Chip Identification (ECID).

The authorization server checks the presented list of measurements against versions whose 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 helps ensure that the update takes place exactly as Apple provided.

Cryptographic measurements list
iBoot
kernel
operating system image
anti-replay value
ECID

Apple Authorization server

Signed authorization response
signature for software
anti-replay value
ECID

How Apple devices interact with the Apple Authorization server.

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 are designed to ensure that, on devices that support personalization, the authorization is for a specific device and that an older operating system or firmware version from one device can't be copied to another. The nonce helps prevent an attacker from saving the server's response and using it to tamper with a device or otherwise alter the system software.

The personalization process is why a network connection to Apple is always required to update any device with Apple-designed silicon, including an Intel-based Mac with the Apple T2 Security Chip.

Finally, the user's data volume is never mounted during a software update, to help prevent anything being read from or written to that volume during updates.

On devices with the Secure Enclave, that hardware similarly uses system software authorization to check the integrity of its software and is designed to prevent downgrade installations.

Operating system integrity

Apple's operating system software is designed with security at its core. This design includes a hardware root of trust—leveraged to enable secure boot—and a secure software update process that's quick and safe. Apple's operating systems also use their purpose-built silicon-based hardware capabilities to help prevent exploitation as the system runs. These runtime features protect the integrity of trusted code while it is being executed. In short, Apple's operating system software helps mitigate attack and exploit techniques—whether those originate from a malicious app, from the web, or through any other channel. Protections listed here are available on devices with supported Apple-designed SoCs, including iOS, iPadOS, tvOS, watchOS, and now macOS on a Mac with Apple silicon.

Feature	A10	A11, S3	A12, S4	A13, S5	A14, S6	M1
Kernel Integrity Protection	✅	✅	✅	✅	✅	✅
Fast Permission Restrictions		✅	✅	✅	✅	✅
System Coprocessor Integrity Protection			✅	✅	✅	✅
Pointer Authentication Codes			✅	✅	✅	✅
Page Protection Layer	✅	✅	✅	✅		See Note below.

Note: Page Protection Layer (PPL) requires that the platform execute *only* signed and trusted code; this is a security model that isn't applicable on macOS.

Kernel Integrity Protection

After the operating system kernel completes initialization, Kernel Integrity Protection (KIP) is enabled to help 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 startup is complete, the memory controller denies writes to the protected physical memory region. The Application Processor's Memory Management Unit (MMU) is configured to help prevent mapping privileged code from physical memory outside the protected memory region and to help 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.

Fast Permission Restrictions

Starting with the Apple A11 Bionic and S3 SoCs, a new hardware primitive was introduced. This primitive, Fast Permission Restrictions, includes a CPU register that quickly restricts permissions per thread. With Fast Permission Restrictions (also known as APRR registers), supported operating systems can remove execute permissions from memory without the overhead of a system call and a page table walk or flush. These registers provide one more level of mitigation for attacks from the web, particularly for code compiled at runtime (just-in-time compiled)—because memory can't be effectively executed at the same time it's being read from and written to.

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)*.

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

Pointer Authentication Codes

Pointer Authentication Codes (PACs) are used to protect against exploitation of memory corruption bugs. System software and built-in apps use PAC to help 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

Item	Key	Salt
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 help ensure a functioning pointer address. Failure to verify results in an abort. 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.

Page Protection Layer

Page Protection Layer (PPL) in iOS, iPadOS, and watchOS is designed to prevent user space code from being modified after code signature verification is complete. Building on Kernel Integrity Protection and Fast Permission Restrictions, PPL manages 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. This protection isn't offered in macOS because PPL is only applicable on systems where all executed code must be signed.

Additional macOS system security capabilities

Additional macOS system security capabilities

macOS operates on a broader set of hardware (for example, Intel-based CPUs, Intel-based CPUs in combination with the Apple T2 Security Chip, and Apple silicon-based SoCs) and supports a range of general-purpose computing use cases. Whereas some users use only the basic preinstalled apps or those available from the App Store, others are kernel hackers who need to disable essentially all platform protections so they can run and test their executing code as with the highest levels of trust. Most fall somewhere between, and many of those have peripherals and software that require varying levels of access. Apple designed the macOS platform with an integrated approach to hardware, software, and services—a platform that provides security by design and makes it simple to configure, deploy, and manage but that retains the configurability that users expect. macOS also includes the key security technologies that an IT professional needs to help protect corporate data and integrate within secure enterprise networking environments.

The following capabilities support and help secure the varied needs of macOS users. They include:

- Signed system volume security

- System Integrity Protection

- Trust caches

- Protection for peripherals

- Rosetta 2 (automatic translation) support and security for a Mac with Apple silicon

- DMA support and protections

- Kernel extension (kext) support and security

- Option ROM support and security

- UEFI firmware security for Intel-based Mac computers

Signed system volume security in macOS

For macOS 10.15, Apple introduced the read-only system volume, a dedicated, isolated volume for system content. macOS 11 adds strong cryptographic protections to system content with a *signed system volume (SSV)*. SSV features a kernel mechanism that verifies the integrity of the system content at runtime and rejects any data—code and noncode—without a valid cryptographic signature from Apple.

SSV not only helps prevent tampering with any Apple software that's part of the operating system, it also makes macOS software update more reliable and much safer. And because SSV uses APFS (Apple File System) snapshots, if an update can't be performed, the old system version can be restored without reinstallation.

Since its introduction, APFS has provided file-system metadata integrity using noncryptographic checksums on the internal storage device. SSV strengthens the integrity mechanism by adding cryptographic hashes, thus extending it to encompass every byte of file data. Data from the internal storage device (including file system metadata) is cryptographically hashed in the read path, and the hash is then compared with an expected value in the file-system metadata. In case of mismatch, the system assumes the data has been tampered with and won't return it to the requesting software.

Each SSV SHA256 hash is stored in the main file-system metadata tree, which is itself hashed. And because each node of the tree recursively verifies the integrity of the hashes of its children—similar to a binary hash (Merkle) tree—the root node's hash value, called a *seal*, therefore encompasses every byte of data in the SSV, which means the cryptographic signature covers the entire system volume.

During macOS installation and update, the seal is recomputed from the file system on-device and that measurement is verified against the measurement which Apple signed. On a Mac with Apple silicon, the bootloader verifies the seal before transferring control to the kernel. On an Intel-based Mac with an Apple T2 Security Chip, the bootloader forwards the measurement and signature to the kernel, which then verifies the seal directly before mounting the root file system. In either case, if the verification fails, the startup process halts and the user is prompted to reinstall macOS.

This procedure is repeated at every boot unless the user has elected to enter a lower security mode and has separately chosen to disable the signed system volume.

SSV and code signing

Code signing is still present and enforced by the kernel. The signed system volume provides protection when any bytes at all are read from the internal storage device. In contrast, code signing provides protection when Mach objects are memory mapped as executable. Both SSV and code signing protect executable code on all read and execute paths.

SSV and FileVault

In macOS 11, equivalent at-rest protection for system content is provided by the SSV, and therefore the system volume no longer needs to be encrypted. Any modifications made to the file system while it's at rest will be detected by the file system when they're read. If the user has enabled FileVault, the user's content on the data volume is still encrypted with a user-provided secret.

If the user chooses to disable the SSV, the system at rest becomes vulnerable to tampering, and this tampering could enable an attacker to extract encrypted user data when the system next starts up. Therefore the system won't permit the user to disable the SSV if FileVault is enabled. Protection while at rest must be enabled or disabled for both volumes in a consistent manner.

In macOS 10.15 or earlier, FileVault protects operating system software while at rest by encrypting user and system content with a key protected by a user-provided secret. This protects against an attacker with physical access to the device from accessing or effectively modifying the file system containing system software.

SSV and a Mac with an Apple T2 Security Chip

On a Mac with an Apple T2 Security Chip, only macOS itself is protected by the SSV. The software that runs on the T2 chip and verifies macOS is protected by secure boot.

System Integrity Protection

macOS utilizes kernel permissions to limit writability of critical system files with a feature called *System Integrity Protection (SIP)*. This feature is separate and in addition to the hardware-based Kernel Integrity Protection (KIP) available on a Mac with Apple silicon, which protects modification of the kernel in memory. Mandatory access control technology is leveraged to provide this and a number of other kernel level protections, including sandboxing and Data Vault.

Mandatory access controls

macOS 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

System Integrity Protection restricts components to read-only in specific critical file system locations to help 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. On an Intel-based Mac, 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.

Trust caches

One of the objects included in the Secure Boot chain is the static trust cache, a trusted record of all the Mach-O binaries that are mastered into the signed system volume. Each Mach-O is represented by a code directory hash. For efficient searching, these hashes are sorted before being inserted into the trust cache. The code directory is the result of the signing operation performed by codesign(1). To enforce the trust cache, SIP must remain enabled. To disable trust cache enforcement on a Mac with Apple silicon, secure boot must be configured to Permissive Security.

When a binary is executed (whether as part of spawning a new process or mapping executable code into an existing process), its code directory is extracted and hashed. If the resulting hash is found in the trust cache, the executable mappings created for the binary will be granted platform privileges—that is, they may possess any entitlement and execute without further verification as to the authenticity of the signature. This is in contrast to an Intel-based Mac, where platform privileges are conveyed to operating system content by the Apple certificate that signs the binaries. (This certificate doesn't constrain which entitlements the binary may possess.)

Nonplatform binaries (for example, notarized third-party code) must have valid certificate chains in order to execute, and the entitlements they may possess are constrained by the signing profile issued to the developer by the Apple Developer Program.

All binaries shipped within macOS are signed with a *platform identifier*. On a Mac with Apple silicon, this identifier is used to indicate that even though the binary is signed by Apple, its code directory hash must be present in the trust cache in order to execute. On an Intel-based Mac, the platform identifier is used to perform targeted revocation of binaries from an older release of macOS; this targeted revocation helps prevent those binaries from executing on newer versions.

The static trust cache completely locks a set of binaries to a given version of macOS. This behavior helps prevent legitimately Apple-signed binaries from older operating systems from being introduced into newer ones in order for an attacker to gain advantage.

Platform code shipped outside the operating system

Apple ships some binaries—for example, Xcode and the development tools stack—that aren't signed with a platform identifier. Even so, they're still permitted to execute with platform privileges on a Mac with Apple silicon and those with a T2 chip. Because this platform software is shipped independently of macOS, it isn't subject to the revocation behaviors imposed by the static trust cache.

Loadable trust caches

Apple ships certain software packages with *loadable trust caches*. These caches have the same data structure as the static trust cache. But although there's only one static trust cache—and its contents are always guaranteed to be locked into read-only ranges after the kernel's early initialization is complete—loadable trust caches are added to the system at runtime.

These trust caches are authenticated either through the same mechanism that authenticates boot firmware (personalization using the Apple trusted signing service) or as globally signed objects (whose signatures don't bind them to a particular device).

One example of a personalized trust cache is the cache, shipped with the disk image that's used to perform field diagnostics on a Mac with Apple silicon. This trust cache is personalized, along with the disk image, and loaded into the subject Mac computer's kernel while it's booted into a diagnostic mode. The trust cache enables the software within the disk image to run with platform privilege.

An example of a globally signed trust cache is shipped with macOS software updates. This trust cache permits a chunk of code within the software update—the *update brain*—to run with platform privilege. The update brain performs any work to stage the software update that the host system lacks the capacity to perform in a consistent fashion across versions.

Peripheral processor security in Mac computers

All modern computing systems have many built-in peripheral processors dedicated to tasks such as networking, graphics, power management, and more. These peripheral processors are often single-purpose and are much less powerful than the primary CPU. Built-in peripherals that don't implement sufficient security become an easier target for attackers to exploit, through which they can persistently infect the operating system. Having infected a peripheral processor firmware, an attacker could target software on the primary CPU or directly capture sensitive data (For example, an Ethernet device could see the contents of packets that aren't encrypted.)

Whenever possible, Apple works to reduce the number of peripheral processors necessary and to avoid designs that require firmware. But when separate processors with their own firmware are required, efforts are taken to help ensure an attacker can't persist on that processor. This can be by verifying the processor in one of two ways:

- Running the processor so that it downloads verified firmware from the primary CPU on startup
- Having the peripheral processor implement its own secure boot chain, to verify the peripheral processor firmware every time the Mac starts up

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

- Ensuring minimum cryptographic strengths
- Ensuring 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 Apple silicon.

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.

Rosetta 2 on a Mac with Apple silicon

A Mac with Apple silicon is capable of running code compiled for the x86_64 instruction set using a translation mechanism called *Rosetta 2*. There are two types of translation offered: just in time and ahead of time.

Just-in-time translation

In the just-in-time (JIT) translation pipeline, an x86_64 Mach object is identified early in the image execution path. When these images are encountered, the kernel transfers control to a special Rosetta translation stub rather than to the dynamic link editor, dyld(1). The translation stub then translates x86_64 pages during the image's execution. This translation takes place entirely within the process. The kernel still verifies the code hashes of each x86_64 page against the code signature attached to the binary as the page is faulted in. In the event of a hash mismatch, the kernel enforces the remediation policy appropriate for that process.

Ahead-of-time translation

In the ahead-of-time (AOT) translation path, x86_64 binaries are read from storage at times the system deems optimal for responsiveness of that code. The translated artifacts are written to storage as a special type of Mach object file. That file is similar to an executable image, but it's marked to indicate it's the translated product of another image.

In this model, the AOT artifact derives all of its identity information from the original x86_64 executable image. To enforce this binding, a privileged userspace entity signs the translation artifact using a device-specific key that's managed by the Secure Enclave. This key is released only to the privileged userspace entity, which is identified as such using a restricted entitlement. The code directory created for the translation artifact includes the code directory hash of the original x86_64 executable image. The signature on the translation artifact itself is known as the *supplemental signature*.

The AOT pipeline begins similarly to the JIT pipeline, with the kernel transferring control to the Rosetta runtime rather than to the dynamic link editor, dyld(1). But the Rosetta runtime then sends an interprocess communication (IPC) query to the Rosetta system service, which asks whether there's an AOT translation available for the current executable image. If found, the Rosetta service provides a handle to that translation, and it's mapped into the process and executed. During execution, the kernel enforces the code directory hashes of the translation artifact which are authenticated by the signature rooted in the device-specific signing key. The original x86_64 image's code directory hashes aren't involved in this process.

Translated artifacts are stored in a Data Vault which isn't runtime-accessible by any entity except for the Rosetta service. The Rosetta service manages access to its cache by distributing read-only file descriptors to individual translation artifacts; this limits access to the AOT artifact cache. This service's interprocess communication and dependent footprint are kept intentionally very narrow to limit its attack surface.

If the code directory hash of the original x86_64 image doesn't match with the one encoded into the AOT translation artifact's signature, this result is considered the equivalent of an invalid code signature, and appropriate enforcement action is taken.

If a remote process queries the kernel for the entitlements or other code identity properties of an AOT-translated executable, the identity properties of the original x86_64 image are returned to it.

Static trust cache content

macOS 11 or later ships with Mach "fat" binaries that contain slices of x86_64 and arm64 computer code. On a Mac with Apple silicon, the user may decide to execute the x86_64 slice of a system binary through the Rosetta pipeline—for example to load a plug-in that has no native arm64 variant. To support this approach, the static trust cache that ships with macOS, generally, contains three code directory hashes per Mach object file:

- A code directory hash of the arm64 slice
- A code directory hash of the x86_64 slice
- A code directory hash of the AOT translation of the x86_64 slice

The Rosetta AOT translation procedure is deterministic in that it reproduces identical output for any given input, irrespective of when the translation was performed or on what device it was performed.

During the macOS build, every Mach object file is run through the Rosetta AOT translation pipeline associated with the version of macOS being built, and the resulting code directory hash is recorded into the trust cache. For efficiency, the actual translated products don't ship with the operating system and are reconstituted on demand when the user requests them.

When an x86_64 image is being executed on a Mac with Apple silicon, if that image's code directory hash is in the static trust cache, the resulting AOT artifact's code directory hash is *also* expected to be in the static trust cache. Such products aren't signed by the device-specific key, because the signing authority is rooted in the Apple secure boot chain.

Unsigned x86_64 code

A Mac with Apple silicon doesn't permit native arm64 code to execute unless a valid signature is attached. This signature can be as simple as an ad hoc code signature (cf. `codesign(1)`) that doesn't bear any actual identity from the secret half of an asymmetric key pair (it's simply an unauthenticated measurement of the binary).

For binary compatibility, translated x86_64 code is permitted to execute through Rosetta with no signature information at all. No specific identity is conveyed to this code through the device-specific Secure Enclave signing procedure, and it executes with precisely the same limitations that native unsigned code executing on an Intel-based Mac.

Direct memory access protections for Mac computers

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 CPU. Since 2012, Mac computers have implemented numerous technologies to protect DMA, resulting in the best and most comprehensive set of DMA protections on any PC.

Direct memory access protections for a Mac with Apple silicon

Apple systems on chip contain an Input/Output Memory Management Unit (IOMMU) for each DMA agent in the system, including PCIe and Thunderbolt ports. Because each IOMMU has its own set of address translation tables to translate DMA requests, peripherals connected by PCIe or Thunderbolt can access only memory that has been explicitly mapped for their use. Peripherals can't access memory belonging to other parts of the system—such as the kernel or firmware—memory assigned to other peripherals. If an IOMMU detects an attempt by a peripheral to access memory that isn't mapped for that peripheral's use, it triggers a kernel panic.

Direct memory access protections for an Intel-based Mac

Intel-based Mac computers with Intel Virtualization Technology for Directed I/O (VT-d) initialize the IOMMU, enabling DMA remapping and interrupt remapping very early in the boot process to mitigate various classes of security vulnerabilities. The Apple IOMMU hardware begins operation with a default-deny policy, so the instant when the system is powered on, it automatically begins blocking DMA requests from peripherals. After being initialized by software, the IOMMUs begin allowing DMA requests from peripherals to memory regions that have been explicitly mapped for their use.

Note: Interrupt remapping for PCIe isn't necessary on a Mac with Apple silicon, because each IOMMU handles MSIs for its own peripherals.

Starting in macOS 11, all Mac computers with an Apple T2 Security Chip run UEFI drivers that facilitate DMA in a restricted ring 3 environment when these drivers are pairing with external devices. This property helps mitigate security vulnerabilities that may occur when a malicious device interacts with a UEFI driver in an unexpected way at boot time. In particular, it reduces the impact of vulnerabilities in a drivers handling of DMA buffers.

Kernel extensions in macOS

Starting with macOS 11, if third-party kernel extensions (kexts) are enabled, they can't be loaded into the kernel on demand. Instead, they're merged into an *Auxiliary Kernel Collection (AuxKC)*, which is loaded during the boot process. For a Mac with Apple silicon, the measurement of the AuxKC is signed into the LocalPolicy (for previous hardware, the AuxKC resided on the data volume). Rebuilding the AuxKC requires the user's approval and restarting of the macOS to load the changes into the kernel, and it requires that the secure boot be configured to Reduced Security.

Important: Kexts are no longer recommended for macOS. Kexts risk the integrity and reliability of the operating system, and Apple recommends users select solutions that don't require extending the kernel.

Kernel extensions in a Mac with Apple silicon

Kexts must be explicitly enabled for a Mac with Apple silicon by holding the power button at startup to enter into One True Recovery (1TR) mode, then downgrading to Reduced Security and checking the box to enable kernel extensions. This action also requires entering an administrator password to authorize the downgrade. The combination of the 1TR and password requirement makes it difficult for software-only attackers starting from within macOS to inject kexts into macOS, which they can then exploit to gain kernel privileges.

After a user authorizes kexts to load, the above User-Approved Kernel Extension Loading flow is used to authorize the installation of kexts. The authorization used for the above flow is also used to capture an SHA384 hash of the user-authorized kext list (UAKL) in the LocalPolicy. The kernel management daemon (kmd) is then responsible for validating only those kexts found in the UAKL for inclusion into the AuxKC.

- If System Integrity Protection (SIP) is enabled, the signature of each kext is verified before being included in the AuxKC.

- If SIP is disabled, the kext signature isn't enforced.

This approach allows Permissive Security flows for developers or users who aren't part of the Apple Developer Program to test kexts before they are signed.

After the AuxKC is created, its measurement is sent to the Secure Enclave to be signed and included in an Image4 data structure that can be evaluated by iBoot at startup. As part of the AuxKC construction, a kext receipt is also generated. This receipt contains the list of kexts that were actually included in the AuxKC, because the set could be a subset of the UAKL if banned kexts were encountered. An SHA384 hash of the AuxKC Image4 data structure and the kext receipt are included in the LocalPolicy. The AuxKC Image4 hash is used for extra verification by iBoot at startup to help ensure that it isn't possible to start up an older Secure Enclave–signed AuxKC Image4 file with a newer LocalPolicy. The kext receipt is used by subsystems such as Apple Pay to determine whether there are any kexts currently loaded that could interfere with the trustworthiness of macOS. If there are, then Apple Pay capabilities may be disabled.

Alternatives to kexts (macOS 10.15 or later)

macOS 10.15 enables 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. Even though 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 road map to transition to them and away from kernel extensions.

User-Approved Kernel Extension Loading

To improve security, user consent is required to load kernel extensions installed with or after installing macOS 10.13. This process 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 a Mac when running macOS 10.12 or earlier

- Are replacing previously approved extensions

- Are allowed to load without user consent by using the `spctl` command-line tool available when a Mac was booted from recoveryOS

- 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.

Option ROM security in macOS

Note: Option ROMs aren't currently supported on a Mac with Apple silicon.

### Option ROM security in a Mac with the Apple T2 Security Chip

Both Thunderbolt and PCIe devices can have an "Option ROM (OROM)" physically attached to the device. (This is typically not a true ROM but is 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 startup phases—for example, to start up from external RAID arrays.

However, because OROMs are generally rewritable, if an attacker overwrites the OROM of a legitimate peripheral, the attacker's code executes early in the boot process and is able to tamper with the execution environment and violate the integrity of software that's loaded later. Likewise, if the attacker introduces their own malicious device to the system, they're also 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 nondefault option can be configured using the `firmwarepasswd` command-line tool in macOS.

OROM sandbox security

In macOS 10.15, UEFI firmware was updated to contain a mechanism for sandboxing OROMs and for stripping privileges from them. UEFI firmware typically executes all code, including OROMs, at the maximum CPU privilege level, called ring 0, and has a single shared virtual memory space for all code and data. Ring 0 is the privilege level where 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.
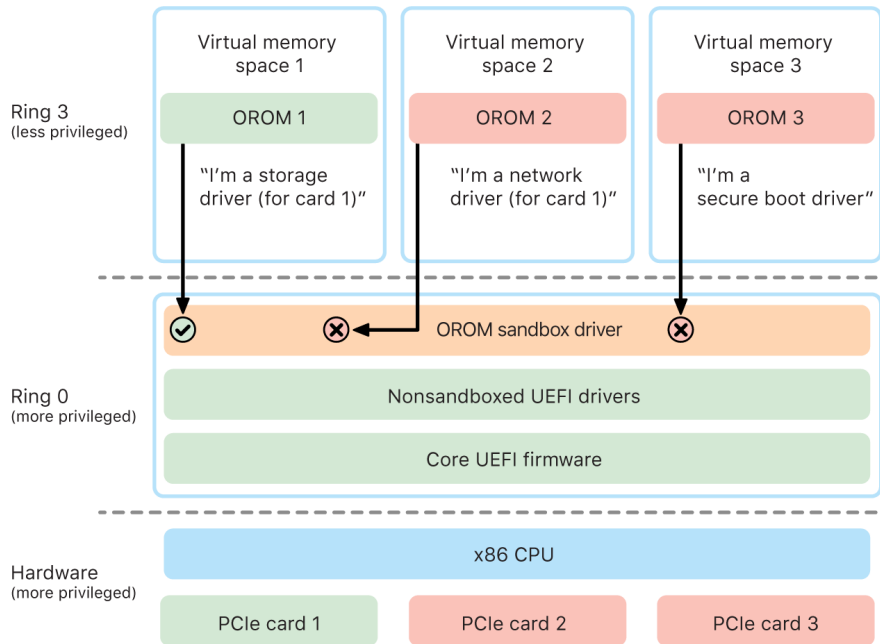
Option ROM (OROM) sandboxing.

The sandbox further significantly restricts both the interfaces that the OROMs can call (much like system call filtering in kernels) and the type of device that an OROM can register as (much like app approval.) The benefit of this design is that malicious OROMs can no longer directly write anywhere within ring 0 memory. Instead, they are 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.

UEFI firmware security in an Intel-based Mac

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.

For an Intel-based Mac 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 help prevent rollback attacks, updates must always have a version newer than the existing one. However, an attacker with physical access to the Mac could potentially 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 and present in all Intel-based Mac computers without the T2 chip.

To help prevent physical attacks that subvert UEFI firmware, Mac computers were rearchitected to root the trust in the UEFI firmware in the T2 chip. On these Mac computers, the root of trust for the UEFI firmware is specifically the T2 firmware, as described in Boot process for an Intel-based Mac.

Intel Management Engine (ME) subcomponent

One subcomponent stored within the UEFI firmware is the *Intel Management Engine (ME)* firmware. The ME—a separate processor and subsystem within Intel chips—is used primarily for audio and video copyright protection on a Mac that has only Intel-based graphics. To reduce this subcomponent's attack surface, an Intel-based Mac runs a custom ME firmware from which most components have been removed. Because the resulting Mac ME firmware is smaller than the default minimal build that Intel makes available, many components that have been the subject of public attacks by security researchers in the past are no longer present.

System Management Mode (SMM)

Intel processors have a special execution mode that's 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)*. No longer a separate microcontroller, the SMC has been integrated into the T2 chip.

System security for watchOS

Apple Watch uses many of the same hardware-based platform security capabilities that iOS and iPadOS use. For example, Apple Watch:

- Performs secure boot and secure software updates

- Maintains operating system integrity

- Helps protect data—both on the device and when communicating with a paired iPhone or the internet

Supported technologies include those listed in System Security (for example, KIP, SKP, and SCIP) as well as Data Protection, keychain, and network technologies.

Updating system software

Apple Watch can be configured for a system software update that will take place that same night. For more information on how the Apple Watch passcode gets stored and used during the update, see Keybags.

Wrist detection

If wrist detection is enabled, the device locks automatically soon after it's removed from the user's wrist. If wrist detection is disabled, Control Center provides an option for locking Apple Watch. When Apple Watch is locked, Apple Pay can be used only by entering the passcode on the Apple Watch. 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.

Activation Lock

When Find My is turned on on iPhone, its paired Apple Watch can use Activation Lock. Activation Lock makes it harder for anyone to use or sell an Apple Watch that's been lost or stolen. Activation Lock requires the user's Apple ID and password to unpair, erase, or reactivate an Apple Watch.

Secure pairing with iPhone

Apple Watch can be paired with only one iPhone at a time. When Apple Watch is unpaired, iPhone communicates instructions to erase all content and data from the watch.

Pairing Apple Watch with iPhone is secured using an out-of-band 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's used for BLE 4.1 out-of-band pairing. Standard BLE Passkey Entry is used as a fallback pairing method, if necessary.

After the BLE session is established and encrypted using the highest security protocol available in the Bluetooth Core Specification, iPhone and Apple Watch exchange keys using either:

- A process adapted from Apple Identity Service (IDS) as described in the iMessage security overview.

- A key exchange using IKEv2/IPsec. The initial key exchange is authenticated using either the Bluetooth session key (for pairing scenarios) or the IDS keys (for operating system update scenarios). Each device generates a random public and private 256-bit Ed25519 key pair, and during the initial key exchange process, the public keys are exchanged.

Note: The mechanism used for key exchange and encryption varies, depending on which operating system versions are on the iPhone and Apple Watch. iPhone devices running iOS 13 or later when paired with an Apple Watch running watchOS 6 or later use only IKEv2/IPsec for key exchange and encryption.

After keys have been exchanged:

- The Bluetooth session key is discarded and all communications between iPhone and Apple Watch are encrypted using one of the methods listed above—with the encrypted Bluetooth, Wi-Fi, and cellular links providing a secondary encryption layer.

- (IKEv2/IPsec only) The keys are stored in the system keychain and used for authenticating future IKEv2/IPsec sessions between the devices. Further communication between these devices is encrypted and integrity protected using ChaCha20-Poly1305 (256-bit keys).

The Bluetooth Low Energy device address is rotated at 15-minute intervals to reduce the risk of the device being locally tracked if someone broadcasts a persistent identifier.

To support apps that need streaming data, encryption is provided with methods described in FaceTime security, using either the Apple Identity Service (IDS) 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. Access-controlled keybags for keychain items are also used. Keys used to communicate between Apple Watch and iPhone are also secured using class-based protection. For more information, see Keybags for Data Protection.

Auto Unlock and Apple Watch

For greater convenience when using multiple Apple devices, some devices can automatically unlock others in certain situations. Auto Unlock supports three uses:

- An Apple Watch can be unlocked by an iPhone.

- A Mac can be unlocked by an Apple Watch.

- An iPhone can be unlocked by an Apple Watch when a user is detected with their nose and mouth covered.

All three use cases are built upon the same basic foundation: a mutually authenticated Station-to-Station (STS) protocol, with Long-Term Keys exchanged at time of feature enablement and unique ephemeral session keys negotiated for each request. Regardless of the underlying communication channel, the STS tunnel is negotiated directly between the Secure Enclaves in both devices, and all cryptographic material is kept within that secure domain (with the exception of Mac computers without a Secure Enclave, which terminate the STS tunnel in the kernel).

Unlocking

A complete unlock sequence can be broken down in two phases. First, the device being unlocked (the "target") generates a cryptographic unlock secret and sends it to the device performing the unlock (the "initiator"). Later, the initiator performs the unlock using the previously generated secret.

To arm auto unlock, the devices connect to each other using a BLE connection. Then a 32-byte unlock secret randomly generated by the target device is sent to the initiator over the STS tunnel. During the next biometric or passcode unlock, the target device wraps its passcode-derived key (PDK) with the unlock secret and discards the unlock secret from its memory.

To perform the unlock, the devices initiate a new BLE connection and then use peer-to-peer Wi-Fi to securely approximate the distance between each other. If the devices are within the specified range and the required security policies are met, the initiator sends its unlock secret to the target through the STS tunnel. The target then generates a new 32-byte unlock secret and returns it to the initiator. If the current unlock secret sent by the initiator successfully decrypts the unlock record, the target device is unlocked and the PDK is rewrapped with the a new unlock secret. Finally, the new unlock secret and PDK are then discarded from the target's memory.

Apple Watch Auto Unlock security policies

For added convenience, Apple Watch can be unlocked by an iPhone directly after initial startup without requiring the user to first enter the passcode on the Apple Watch itself. To achieve this, the random unlock secret (generated during the very first unlock sequence after enablement of the feature) is used to create a long-term escrow record, which is stored in the Apple Watch keybag. The escrow record secret is stored in the iPhone keychain and used to bootstrap a new session after each Apple Watch restart.

iPhone Auto Unlock security policies

Additional security policies apply to iPhone Auto Unlock with Apple Watch. Apple Watch can't be used in place of Face ID on iPhone for other operations, such as Apple Pay or app authorizations. When Apple Watch successfully unlocks a paired iPhone, the watch displays a notification and plays an associated haptic. If the user taps the Lock iPhone button in the notification, the watch sends the iPhone a lock command over BLE. When the iPhone receives the lock command, it locks and disables both Face ID and unlock using Apple Watch. The next iPhone unlock must be performed with the iPhone passcode.

Successfully unlocking a paired iPhone from Apple Watch (when enabled) requires that the following criteria be met:

- iPhone must have been unlocked using another method at least once after the associated Apple Watch was placed on wrist and unlocked.

- Sensors must be able to detect that the nose and mouth are covered.

- Distance measured must be 2–3 meters or less

- Apple Watch must not be in bedtime mode.

- Apple Watch or iPhone must have been unlocked recently, or Apple Watch must have experienced physical motion indicating that the wearer is active (for example, not asleep).

- iPhone must have been unlocked at least once in the past 6.5 hours.

- iPhone must be in a state where Face ID is allowed to perform a device unlock. (For more information, see Touch ID, Face ID, passcodes, and passwords.)

Approve in macOS 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

Secure use of Wi-Fi, cellular, iCloud, and Gmail

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 the 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 must 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 that it can be used to fetch Mail. This OAuth token is never used for connectivity with the Gmail server from the paired iPhone.

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 hardware TRNG
- Timing-based jitter collected during boot
- Entropy collected from hardware interrupts
- A seed file used to persist entropy across boots
- Intel random instructions—for example, RDSEED and RDRAND (only on an Intel-based Mac)

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 (/dev/random)

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

Apple Security Research Device

The Apple Security Research Device is a specially fused iPhone that allows security researchers to perform research on iOS without having to defeat or disable the platform security features of iPhone. With this device, a researcher can side-load content that runs with platform-equivalent permissions and thus perform research on a platform that more closely models that of production devices.

To help ensure that user devices aren't affected by the security research device execution policy, the policy changes are implemented in a variant of iBoot and in the Boot Kernel Collection. These fail to boot on user hardware. The research iBoot checks for a new fusing state and enters a panic loop if it's being run on nonresearch fused hardware.

The cryptex subsystem allows a researcher to load a personalized trust cache and a disk image containing corresponding content. A number of defense in-depth measures have been implemented that are designed to ensure that this subsystem doesn't allow execution on user devices:

- `launchd` won't load the `cryptexd` launchd property list if it's unable to detect the research fuse.

- `cryptexd` aborts if it doesn't detect the research fuse.

- The entitlement that grants `cryptexd` the ability to mount a disk image is honored only by the research kernel cache. The relevant code path isn't compiled into the release kernel cache.

- The signing server refuses to personalize a cryptex disk image for a device not on an explicit allow list.

To respect the privacy of the security researcher, only the measurements (for example, hashes) of the executables and the security research device identifiers are sent to Apple during personalization. Apple doesn't receive the content of the cryptex being loaded onto the device.

To avoid having a malicious party attempt to masquerade a research device as a user device to trick a target into using it for everyday usage, the security research device has the following differences:

- The security research device starts up only while charging. This can be using a Lightning cable or a Qi-compatible charger. If the device isn't charging during startup, the device enters Recovery mode. If the user starts charging and restarts the device, it starts up as normal. As soon as XNU starts, the device doesn't need to be charging to continue operation.

- The words *Security Research Device* are displayed below the Apple logo during iBoot startup.

- The XNU kernel boots in verbose mode.

- The device is etched on the side with the message "Property of Apple. Confidential and Proprietary. Call +1 877 595 1125."

The following are additional measures that are implemented in software that appears after boot:

- The words *Security Research Device* are displayed during device setup.

- The words *Security Research Device* are displayed on the lock screen and in the Settings app.

The Security Research Device affords researchers the following abilities that a user device doesn't:

- Side-load executable code onto the device with arbitrary entitlements at the same permission level as Apple operating system components.

- Start services at startup.

- Persist content across restarts.

Encryption and Data Protection

Encryption and Data Protection overview

The secure boot chain, system security, and app security capabilities all help to verify 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 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*, whereas the data on an Intel-based Mac is protected with a volume encryption technology called *FileVault*. A Mac with Apple silicon uses a hybrid model that supports Data Protection, with two caveats: The lowest protection level Class (D) isn't supported, and the default level (Class C) uses a volume key and acts just like the FileVault on an Intel-based Mac. In all cases, key management hierarchies are rooted in the dedicated silicon of the Secure Enclave, and a dedicated AES Engine supports line-speed encryption and helps ensure that long-lived encryption keys aren't exposed to the kernel operating system or CPU (where they might be compromised). (An Intel-based Mac with a T1 or lacking a Secure Enclave doesn't use dedicated silicon to protect its FileVault encryption keys.)

Besides using Data Protection and FileVault to help prevent unauthorized access to data, Apple uses *operating system kernels* to enforce protection and security. The kernel uses access controls to sandbox apps (which restricts what data an app can access) and a mechanism called a *Data Vault* (which rather than restricting the calls an app can make, restricts access to the data of an app from all other requesting apps).

Passcodes and passwords

Passcodes in devices that support Data Protection

By setting up a device passcode or password, the user automatically enables Data Protection. iOS and iPadOS support six-digit, four-digit, and arbitrary-length alphanumeric passcodes. Besides unlocking the device, a passcode or password 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 or password 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. In fact, 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. And by using Touch ID and Face ID, the user can establish a much stronger passcode than would otherwise be practical. The stronger passcode increases the effective amount of entropy protecting the encryption keys used for Data Protection, without adversely affecting the user experience of unlocking a 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.

Delays between passcode attempts

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

If the Erase Data option is turned on (in Settings > Touch ID & Passcode), after 10 consecutive incorrect attempts to enter the passcode, all content and settings are removed from storage. 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 through Microsoft Exchange ActiveSync, and can be set to a lower threshold.

On devices with Secure Enclave, the delays are enforced by the Secure Enclave. 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 > Touch ID & Passcode or Face ID & Passcode.

Delays between passwords attempts in macOS

To help prevent brute-force attacks, when Mac starts up, no more than 10 password attempts are allowed at the Login Window or using Target Disk Mode, and escalating time delays are imposed after a certain number of incorrect attempts. The delays are enforced by the Secure Enclave. If Mac is restarted during a timed delay, the delay is still enforced, with the timer starting over for the current period.

To help prevent malware from causing permanent data loss by trying to attack the user's password, these limits aren't enforced after the user has successfully logged in to the Mac, but they are reimposed after reboot. If the 10 attempts are exhausted, 10 more attempts are available after booting into recoveryOS. And if those are also exhausted, then 10 additional attempts are available for each FileVault recovery mechanism (iCloud recovery, FileVault recovery key, and institutional key), for a maximum of 30 additional attempts. After 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 an MDM solution. 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 on a Mac with Apple silicon and those with the T2 chip

Attempts	Delay enforced
5	1 minute
6	5 minutes
7	15 minutes
8	15 minutes
9	1 hour
10	Disabled

In a Mac with the Apple T2 Security Chip, the password serves a similar function 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

Data Protection

Data Protection overview

Apple uses a technology called Data Protection to protect data stored in flash storage on the devices that feature an Apple SoC such as iPhone, iPad, Apple Watch, Apple TV, and a Mac with Apple silicon. 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. Certain system apps (such as Messages, Mail, Calendar, Contacts, Photos) and Health data values use Data Protection by default. Third-party apps 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 Apple devices. 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. APFS (Apple File System) allows the file system to further subdivide the keys into a per-extent basis (where portions of a file can have different keys).

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. On A14 and M1 devices, the encryption uses AES-256 in XTS mode where the 256-bit per-file-key goes through a Key Derivation Function (NIST Special Publication 800-108) to derive a 256-bit tweak and a 256-bit cipher key. The hardware generations of A9 through A13, S5, and S6 use AES-128 in XTS mode where the 256-bit per file key was split to provide a 128-bit tweak and a 128-bit cipher key.

On a Mac with Apple silicon, Data Protection defaults to Class C (see Data Protection classes), but utilizes a volume key rather than a per-extent or per-file key—effectively re-creating the security model of FileVault for user data. Users must still opt-in to FileVault in order to receive the full protection of entangling the encryption key hierarchy with their password. Developers can also opt in to a higher protection class that uses a per-file or per-extent key.

Data Protection in Apple devices

On Apple devices with Data Protection, each file is protected with a unique per-file (or per-extent) key. The key, wrapped using the NIST AED key wrap algorithm, is further wrapped with one of several class keys, depending on how the file is meant to be accessed. The wrapped per-file key is then 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 Application Processor. At startup, the Secure Enclave negotiates an ephemeral key with the AES Engine. When the Secure Enclave unwraps a file's keys, they're 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 the operating system is 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 systems, to achieve effaceability and to protect its key wrapping key, among other assets. For more information, see Secure nonvolatile storage.

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 or using a media key-wrapping key, protected by Secure Enclave anti-replay mechanism. 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 devices supporting Data Protection, 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. Apple silicon-based Mac computers don't support Class D: No Protection, and a security boundary is established around logging in and out (not locking or unlocking as on iPhone, iPad, and iPod touch).

Class	Protection type
Class A: Complete Protection	(NSFileProtectionComplete)
Class B: Protected Unless Open	(NSFileProtectionCompleteUnlessOpen)
Class C: Protected Until First User Authentication *Note:* macOS uses a volume key to recreate FileVault protection characteristics.	(NSFileProtectionCompleteUntilFirstUser Authentication)

Class	Protection type
Class D: No Protection	(NSFileProtectionNone)

Note: Not supported on macOS.

Complete Protection

(NSFileProtectionComplete): The class key is protected with a key derived from the user passcode or password 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 (logs in to) the device using Touch ID or Face ID.

In macOS, shortly after the last user is logged out, the decrypted class key is discarded, rendering all data in this class inaccessible until a user enters the passcode again or logs into the device using Touch ID.

Protected Unless Open

(NSFileProtectionCompleteUnlessOpen): Some files may need to be written while the device is locked or the user is logged out. 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. SHA256 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.

In macOS, the private part of NSFileProtectionCompleteUnlessOpen is accessible as long as any users on the system are logged in or are authenticated.

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 or the user logged out. 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.

In macOS, this class utilizes a volume key which is accessible as long as the volume is mounted, and acts just like FileVault.

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).

This isn't supported in macOS.

Note: In macOS, for volumes that don't correspond to a booted operating system, all data protection classes are accessible as long as the volume is mounted. The default data protection class is *NSFileProtectionCompleteUntilFirstUserAuthentication*. Per-extent key functionality is available to both Rosetta 2 and native apps.

Keybags for Data Protection

The keys for both file and keychain Data Protection classes are collected and managed in keybags on iOS, iPadOS, watchOS, and tvOS. These operating systems 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. 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 later SoCs, the .plist file contains a key that indicates that the keybag is stored in a locker protected by the 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. 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 the Finder (macOS 10.15 or later) or iTunes (in macOS 10.14 or earlier) 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, nonmigratory 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—is run through 10 million iterations of the key derivation function 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 syncing with the Finder (macOS 10.15 or later) or iTunes (in macOS 10.14 or earlier) through USB and mobile device management (MDM). This keybag allows the Finder or 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 the Finder or iTunes, or on the MDM solution that remotely manages the device.

The escrow keybag improves the user experience during device syncing, which potentially requires access to all classes of data. When a passcode-locked device is first connected to the Finder or 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 the Finder or 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're 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 UID.

On A9 (and later) 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. In iOS 13 and iPadOS 13.1 or later, the token is stored in a locker protected by the Secure Enclave. Prior to iOS 13, this token was exported from the Secure Enclave and written to Effaceable Storage or was protected by the Secure Enclave anti-replay mechanism. A policy timer incremented the counter if the device hadn't rebooted within 20 minutes.

Unattended software updates occur when the system detects an update is available and when one of the following is true:

- Automatic updates are configured in iOS 12 or later.
- 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 re-created 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.

Protecting keys in alternate boot modes

Data Protection is designed to provide access to user data only after successful authentication, and only to the authorized user. Data protection classes are designed to support a variety of use cases, such as the ability to read and write some data even when a device is locked (but after first unlock). Additional steps are taken to protect access to user data during alternate boot modes such as those used for Device Firmware Update (DFU) mode, Recovery mode, Apple Diagnostics, or even during software update. These capabilities are based on a combination of hardware and software features, and have been expanded as Apple-designed silicon has evolved.

Feature	A10	A11, S3	A12, S4	A13, S5	A14, M1, S6
Recovery: All Data Protection Classes protected	✅	✅	✅	✅	✅
Alternate boots of DFU mode, Recovery, and software updates: Class A, B, and C data protected			✅	✅	✅

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. How the seed bit is used varies according to the system on chip:

- 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 file system metadata key and Class D keys).

- In iOS 13 or later and iPadOS 13.1 or later on devices with an A10 or later, all user data is rendered cryptographically inaccessible when devices are booted into Diagnostics 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.

- On A12 SoCs, the Secure Enclave Boot ROM locks the passcode seed bit if the Application Processor has entered Device Firmware Upgrade (DFU) mode or Recovery mode. When the passcode seed bit is locked, no operation to change it is allowed. This is designed to prevent access to data protected with 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 on your iPhone, or your iPhone is disabled
Apple TV	Restore your Apple TV
A Mac with Apple silicon	Revive or restore a Mac with Apple silicon

Protecting user data in the face of attack

Attackers attempting to extract user data often try a number of techniques: extracting the encrypted data to another medium for brute-force attack, manipulating the operating system version, or otherwise changing or weakening the security policy of the device to facilitate attack. Attacking data on a device often requires communicating with the device using physical interfaces like Lightning or USB. Apple devices include features to help prevent such attacks.

Apple devices support a technology called *Sealed Key Protection (SKP)* that's designed to ensure that cryptographic material is rendered unavailable off device, or that's used if manipulations are made to operating system versions or security settings without appropriate user authorization. This feature is *not* provided by the Secure Enclave; instead, it's supported by hardware registers that exist at a lower layer in order to provide an additional layer of protection to the keys necessary to decrypt user data independent of the Secure Enclave.

*Note:* SKP is available only on devices with an Apple-designed SoC.

Feature	A10	A11, S3	A12, S4	A13, S5	A14, M1, S6
Sealed Key Protection	✅	✅	✅	✅	✅

iPhone and iPad can also be configured to only activate data connections in conditions more likely to indicate the device is still under the physical control of the authorized owner.

Sealed Key Protection (SKP)

On Apple devices that support Data Protection, the key encryption key (KEK) is protected (or sealed) with measurements of the software on the system, as well as being tied to the UID available only from the Secure Enclave. On a Mac with Apple silicon, the protection of the KEK is further strengthened by incorporating information about security policy on the system, because macOS supports critical security policy changes (for example, disabling secure boot or SIP) that are unsupported on other platforms. On a Mac with Apple silicon, this protection encompass FileVault keys, because FileVault is implemented using Data Protection (Class C).

The key that results from entangling the user password, long-term SKP key, and Hardware key 1 (the UID from Secure Enclave) is called the *password-derived key*. This key is used protect the user keybag (on all supported platforms) and KEK (on macOS only), and then enable biometric unlock or auto unlock with other devices such as Apple Watch.
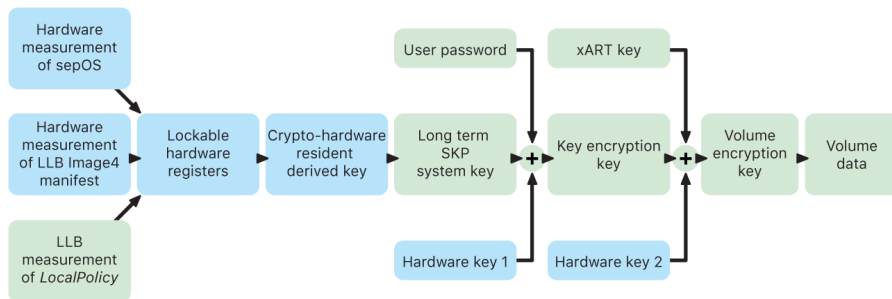
The Sealed Key Protection process in a Mac with Apple silicon.

The Secure Enclave Boot Monitor captures the measurement of the Secure Enclave OS that is loaded. When the Application Processor Boot ROM measures the Image4 manifest attached to LLB, that manifest contains a measurement of all other system-paired firmware that is loaded as well. The LocalPolicy contains the core security configurations for the macOS which are loaded. The LocalPolicy also contains the `nsih` field which is a hash of the macOS Image4 manifest. The macOS Image4 manifest contains measurements of all of the macOS-paired firmware and core macOS boot objects such as the Boot Kernel Collection or signed system volume (SSV) root hash.

If an attacker is able to unexpectedly change any of the above measured firmware, software, or security configuration components, it modifies the measurements stored in the hardware registers. The modification of the measurements causes the crypto-hardware-derived *system measurement root key (SMRK)* to derive to a different value, effectively breaking the seal on the key hierarchy. That causes the *system measurement device key (SMDK)* to be inaccessible, which in turn causes the KEK, and thus the data, to be inaccessible.

However, when the system isn't under attack, it must accommodate legitimate software updates that change the firmware measurements and the nsih field in the LocalPolicy to point at new macOS measurements. In other systems that attempt to incorporate firmware measurements but that don't have a known good source of truth, the user is required to disable the security, update firmware, and then reenable so that a new measurement baseline can be captured. This significantly increases the risk that the attacker could tamper with firmware during a software update. The system is helped by the fact that the Image4 manifest contains all the measurements needed. The hardware that decrypts the SMDK with the SMRK when the measurements match during a normal boot can also encrypt the SMDK to to a proposed future SMRK. By specifying the measurements that are expected after a software update, the hardware can encrypt an SMDK, which is accessible in a current operating system, so that it remains accessible in a future operating system. Similarly, when a customer legitimately changes their security settings in the LocalPolicy, the SMDK must be encrypted to the future SMRK based on the measurement for the LocalPolicy, which LLB computes on the next restart.

Activating data connections securely in iOS and iPadOS

On iOS or iPadOS devices, if no data connection has been established recently, users must use Touch ID, Face ID, or a passcode to activate data connections through a Lightning, USB, or Smart Connector interface. 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:

- Helps ensure that frequent users of connections to a Mac or PC, to accessories, or wired to CarPlay won't need to enter 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 3 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).

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 iPhone, iPad, iPod touch, Mac, Apple TV, and Apple Watch. 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.

Space sharing

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.

Multiple volumes

In macOS 10.15 or later, 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)

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

- *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 volumes are all shared and not duplicated.

In macOS 11, the system volume is captured in a snapshot. The operating system boots from a snapshot of the system volume, not just from a read-only mount of the mutable system volume.

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

- System volume
- Data volume

Keychain data protection

Many apps need to handle passwords and other short but sensitive bits of data, such as keys and login tokens. The keychain provides a secure way to store these items. The various Apple operating systems use differing mechanisms to enforce the guarantees associated with the different keychain protection classes. In macOS (including a Mac with Apple silicon), Data Protection is not used directly to enforce these guarantees.

Overview

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, and the secret value (kSecValueData) is encrypted with the secret key. The metadata 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 round trip 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 be shared only between apps from the same developer. To share keychain items, third-party apps to use access groups with a prefix allocated to them through the Apple Developer Program in their 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 functions.

Availability	File data protection	Keychain data protection
When unlocked	NSFileProtectionComplete	kSecAttrAccessibleWhenUnlocked
While locked	NSFileProtectionCompleteUnless Open	N/A
After first unlock	NSFileProtectionCompleteUntil FirstUserAuthentication	kSecAttrAccessibleAfterFirstUnlock
Always	NSFileProtectionNone	kSecAttrAccessibleAlways
Passcode enabled	N/A	kSecAttrAccessibleWhenPasscode SetThisDeviceOnly

Apps that use 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 "nonmigratory," so it can't be moved to another device.

Keychain data class protections

The class protections listed below are enforced for keychain items.

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
iMessage keys	After first unlock
Home sharing password	When unlocked
Safari passwords	When unlocked
Safari bookmarks	When unlocked
Finder/iTunes backup	When unlocked, nonmigratory
VPN certificates	Always, nonmigratory
Bluetooth® keys	Always, nonmigratory
Apple Push Notification service (APNs) token	Always, nonmigratory

Item	Accessible
iCloud certificates and private key	Always, nonmigratory
Certificates and private keys installed by a configuration profile	Always, nonmigratory
SIM PIN	Always, nonmigratory
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 or password. 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 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.

Keychain architecture in macOS

macOS also provides access to the keychain to conveniently and securely stores user names and passwords, 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 relying on 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.

FileVault

Volume encryption with FileVault in macOS

Mac computers offer 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.

FileVault on a Mac with Apple silicon is implemented using Data Protection Class C with a volume key. On a Mac with the Apple T2 Security Chip as well as a Mac with Apple silicon, encrypted internal storage devices directly connected to the Secure Enclave leverage its hardware security capabilities as well as that of the AES engine. After a user turns on FileVault on a Mac, their credentials are required during the boot process.

Internal storage with FileVault turned on

Without valid login credentials or a cryptographic recovery key, the internal APFS volumes remain encrypted and are protected from unauthorized access even if the physical storage device is removed and connected to another computer. In macOS 10.15, this includes both the system volume and the data volume. Starting in macOS 11, the system volume is protected by the signed system volume (SSV) feature, but the data volume remains protected by encryption. Internal volume encryption on a Mac with Apple silicon as well as those 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
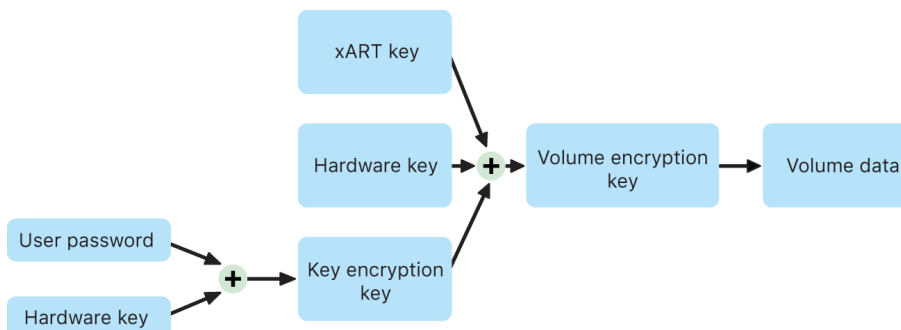
Internal volume encryption when FileVault is turned on in macOS.

On a Mac with Apple silicon and those 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 encryption key by default. Volume and metadata contents are encrypted with this volume encryption 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.

Internal storage with FileVault turned off

If FileVault isn't turned on in a Mac with Apple silicon or a Mac with the T2 chip during the initial Setup Assistant process, the volume is still encrypted but the volume encryption key is protected only by the hardware UID in the Secure Enclave.
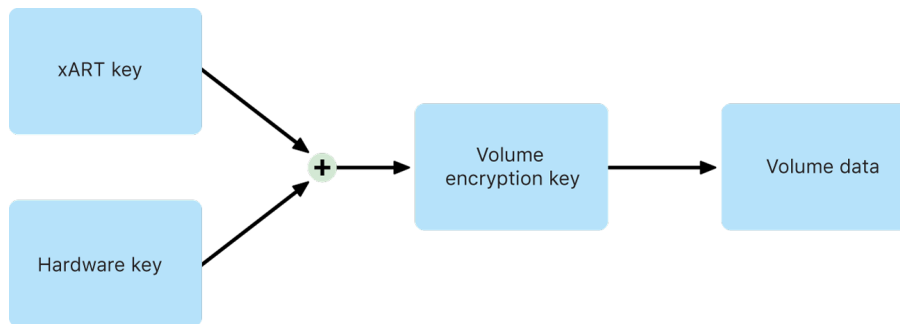
Internal volume encryption when FileVault is turned off in macOS.

If FileVault is turned on later—a process that is immediate because the data has already been encrypted—an anti-replay mechanism helps prevent 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 encryption key is securely deleted by the Secure Enclave. This helps prevent future access with this key even by the Secure Enclave. In addition, all volume encryption keys are wrapped with a media key. The media key doesn't provide additional confidentiality of data; instead, it's designed to enable swift and secure deletion of data because without it decryption is impossible.

On a Mac with Apple silicon and those with the T2 chip, the media key is guaranteed to be erased by the Secure Enclave supported technology—for example by remote MDM commands. Erasing the media key in this manner renders the volume cryptographically inaccessible.

Removable storage devices

Encryption of removable storage devices doesn't utilize the security capabilities of the Secure Enclave, and its encryption is performed in the same manner as an Intel-based Mac without the T2 chip.

Managing FileVault in macOS

Using Secure Token

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, setting the first user's password, or during the first login by a user of the Mac. This implementation of the encryption keys, when they're generated, and how they're stored are all part of a feature known as *Secure Token*. Specifically, a secure token 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 a personal recovery key (PRK) that can be stored with a mobile device management (MDM) solution for escrow

- Create and use an institutional recovery key (IRK)

- Defer enablement of FileVault until a user logs in to or out of the Mac

In macOS 11, setting the initial password for the very first user on the Mac results in that user being granted a secure token. In some workflows, that may not be the desired behavior, as previously, granting the first secure token would have required the user account to log in. To prevent this from happening, add `;DisabledTags;SecureToken` to the programmatically created user's `AuthenticationAuthority` attribute prior to setting the user's password, as shown below:

```
sudo dscl . append /Users/<user name> AuthenticationAuthority
";DisabledTags;SecureToken"
```

Using Bootstrap Token

macOS 10.15 introduced a new feature—Bootstrap Token—to help with granting a secure token to both mobile accounts and the optional device enrollment-created administrator account ("managed administrator"). In macOS 11, the Bootstrap Token can grant a secure token to any user logging in to a Mac computer, including local user accounts. Using the Bootstrap Token feature of macOS 10.15 or later requires:

- Mac enrollment in MDM using Apple School Manager or Apple Business Manager, which makes the Mac supervised

- MDM vendor support

In macOS 10.15.4 or later, a Bootstrap Token is generated and escrowed to MDM on the first login by any user who is Secure Token–enabled if the MDM solution supports the feature. A bootstrap token can also be generated and escrowed to MDM using the `profiles` command-line tool, if needed.

In macOS 11, the bootstrap token may also be used for more than just granting secure token to user accounts. On a Mac with Apple silicon, the bootstrap token, if available, can be used to authorize the installation of both kernel extensions and software updates when managed using MDM.

How Apple protects users' personal data

Protecting app access to user data

In addition to encrypting data at rest, Apple devices help prevent apps from accessing a user's personal information without permission using various technologies including Data Vault. In Settings in iOS and 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

In iOS 13.4 or later and iPadOS 13.4 or later, all third-party apps automatically have their data protected in a Data Vault. Data Vault helps protect against unauthorized access to the data even from processes that aren't themselves sandboxed.

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. iOS and iPadOS also provide restrictions designed to prevent data movement between apps and accounts installed by a mobile device management (MDM) solution and those installed by the user.

Protecting access to user's health data

HealthKit provides a central repository for health and fitness data on iPhone and Apple Watch. 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 the 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.

Collecting and storing health and fitness data

HealthKit also collects and stores 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 the 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 the 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 requires iOS 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 updates 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 the Finder (macOS 10.15 or later) or iTunes (in macOS 10.14 or earlier), health data is stored only if the backup is encrypted.

Clinical health records

Users can sign in to 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 1.3 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 permission to read data, apps can read data written by all sources. However, apps can't determine access granted to other apps. In addition, apps can't conclusively tell whether they've been granted read access to health data. When an app doesn't have read access, all queries return no data—the same response that an empty database would return. This is designed to prevent 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 synced 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 the 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 in iOS 13 or earlier. In iOS 14, Medical ID is synced between devices using CloudKit and has the same encryption characteristics as the rest of health data.

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 a 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 Personal Identification Verification (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 PIV cards. These cards are widely used in commercial and government organizations for two-factor authentication, 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 certificate authority (CA) certificate.

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.

App security

App security overview

Today, apps are among the most critical elements of a security architecture. Even as 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 help 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 in macOS 10.15 or later, all Mac apps need to be notarized by Apple to launch. This requirement helps to ensure that these apps are free of known malware without requiring that the apps be provided through the App Store. In addition, macOS includes state-of-the-art antivirus 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 protected—which helps ensure 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

Native capability	Third-party equivalent
Gatekeeper	Endpoint protection; enforces code signing on apps to help ensure that only trusted software runs
eficheck (Necessary for a Mac without an Apple T2 Security Chip)	Endpoint protection; rootkit detection
Application firewall	Endpoint protection; firewalling
Packet Filter (pf)	Firewall solutions
System Integrity Protection	Built into macOS
Mandatory Access Controls	Built into macOS
Kext exclude list	Built into macOS
Mandatory app code signing	Built into macOS
App notarization	Built into macOS

App security in iOS and iPadOS

App security overview for iOS and iPadOS

Unlike other mobile platforms, iOS and iPadOS don't allow users to install potentially malicious unsigned apps from websites or to run untrusted apps. At runtime, code signature checks that all executable memory pages are made as they are loaded to help 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 in iOS and iPadOS

Mandatory code signing

After the iOS or iPadOS kernel has started, it controls which user processes and apps can be run. To help 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 operating system to apps and helps prevent third-party apps from loading unsigned code resources or using self-modifying code.

How developers sign their apps

Developers can sign their apps through certificate validation (through the Apple Developer Program). They can also embed frameworks inside their apps and have that code validated with an Apple-issued certificate (through a team identifier string).

- *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 help 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.

- *Code signature validation:* 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 helps ensure 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 in iOS and iPadOS

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 is designed to prevent 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 users' apps. Most iOS and iPadOS system files and resources run as the nonprivileged user "mobile," as do all third-party apps. The entire operating system 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.

Address Space Layout Randomization

Address Space Layout Randomization (ASLR) helps protect against the exploitation of memory corruption bugs. Built-in apps use ASLR to help randomize all memory regions upon launch. Randomly arranging the memory addresses of executable code, system libraries, and related programming constructs reduces the likelihood of many 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 more difficult to execute, especially across multiple devices. Xcode, and the iOS or iPadOS development environments, automatically compile third-party programs with ASLR support turned on.

Execute Never feature

Further protection is provided by iOS and iPadOS using ARM's Execute Never (XN) feature, which marks memory pages as nonexecutable. 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 in iOS, iPadOS, and macOS

iOS, iPadOS, and macOS 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 because it's enabled by the user for the entire system. After it's 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 help prevent users from exporting a document from a managed app to an unmanaged Document Provider, or help prevent them from using an unmanaged keyboard with a managed app. Additionally, app developers can prevent the use of third-party keyboard extensions within their app.

App protection and app groups in iOS and iPadOS

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 helps ensure that App group IDs (GIDs) are unique across the app ecosystem.

Verifying accessories in iOS and iPadOS

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 or USB-C connector or through Bluetooth, the device asks the accessory to prove it's 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 or USB-C cable, or location information provided over Bluetooth. An authentication IC is designed to ensure 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 AES128 in counter (CTR) mode. Ephemeral keys are exchanged using ECDH key exchange (Curve25519) and signed using the authentication IC's 1024-bit RSA key as part of the Station-to-Station (STS) protocol.

App security in macOS

App security overview in macOS

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

App code signing process in macOS

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

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

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

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

Gatekeeper and runtime protection in macOS

Gatekeeper

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

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

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

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

Runtime protection

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

Protecting against malware in macOS

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

1. *Prevent launch or execution of malware:* App Store or Gatekeeper and Notarization

2. *Block malware from running on customer systems:* Gatekeeper, Notarization, and XProtect

3. *Remediate malware that has executed:* MRT

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

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

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

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

Notarization

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

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

XProtect

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

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

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

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

Malware Removal Tool

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

Automatic security updates

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

Response process

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

- Any associated Developer ID certificates are revoked.

- Notarization revocation tickets are issued for all files (apps and associated files).

- XProtect signatures are developed and released.

- MRT signatures are developed and released.

- These signatures are also applied retroactively to previously notarized software, and any new detections can result in one or more of the previous actions occurring.

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

Controlling app access to files in macOS

Apple believes that users should have full transparency, consent, and control over what apps are doing with their data. In macOS 10.15, this model is enforced by the system to help ensure that all apps must obtain user consent before accessing files in Documents, Downloads, Desktop, iCloud Drive, and 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 help ensure they don't circumvent other protections. Depending on the access policy, users may be asked to, or required to, change the setting in System Preferences > Security & Privacy > Privacy:

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

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

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

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

Secure features in the Notes app

Secure notes

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

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

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

- The user taps the Lock Now button in Notes

- Notes is switched to the background for more than 3 minutes (8 minutes in macOS)

- The iOS or iPadOS device locks

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

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

Shared notes

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

Secure features in the Shortcuts app

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

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

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

Services security

Services security overview

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

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

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

Apple ID and Managed Apple ID

Apple ID security overview

Overview

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

- Must be at least eight characters in length

- Must contain both letters and numbers

- Must not contain more than three consecutive identical characters

- Can't be a commonly used password

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

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

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

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

Two-factor authentication

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

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

Account recovery

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

Managed Apple ID security

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

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

Inspecting Managed Apple IDs

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

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

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

Managed Apple IDs and personal devices

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

iCloud

iCloud security overview

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

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

iCloud Drive security

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

iCloud Drive backup

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

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

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

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

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

Security of iCloud Backup

The following content is backed up using iCloud Backup:

- Records for purchased music, movies, TV shows, apps, and books. A user's iCloud Backup includes information about purchased content present on the user's device, but not the purchased content itself. When the user restores from an iCloud Backup, their purchased content is automatically downloaded from the iTunes Store, the App Store, the Apple TV app, or Apple Books. Some types of content aren't downloaded automatically in all countries or regions, and previous purchases may be unavailable if they have been refunded or are no longer available in the store. Full purchase history is associated with a user's Apple ID.

- Photos and videos on a user's devices. Note that if a user turns on iCloud Photos in iOS 8.1 or later, iPadOS 13.1 or later, or OS X 10.10.3 or later, their photos and videos are already stored in iCloud, so they aren't included in the user's iCloud Backup.

- Contacts, calendar events, reminders, and notes

- Device settings

- App data

- Home screen and app organization

- HomeKit configuration

- Medical ID data

- Visual Voicemail password (requires the SIM card that was in use during backup)

- iMessage, Business Chat, text (SMS), and MMS messages (requires the SIM card that was in use during backup)

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

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

CloudKit end-to-end encryption

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

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

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

Passcode and password management

Password security overview

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

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

Sign in with Apple security

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

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

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

Automatic strong passwords

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

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

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

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

Password AutoFill security

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

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

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

Password AutoFill in apps

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

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

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

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

App access to saved passwords

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

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

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

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

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

Password security recommendations

Overview

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

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

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

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

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

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

Upgrading account authentication security with Account Authentication Modification Extensions

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

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

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

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

Password Monitoring

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

How monitoring works

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

How passwords are matched

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

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

Comparing users' passwords against a curated list

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

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

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

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

Sending passwords to other users or Apple devices

Saving credentials to another device with AirDrop

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

Filling in credentials in apps on Apple TV

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

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

- If the device and Apple TV uses the same iCloud account, encryption between the devices happens automatically.

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

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

Credential provider extensions

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

iCloud Keychain

iCloud Keychain security overview

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

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

- A user's iCloud account is compromised.

- iCloud is compromised by an external attacker or employee.

- A third party accesses user accounts.

Password manager integration with iCloud Keychain

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

Secure keychain syncing

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

Syncing circle is placed in user's iCloud

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

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

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

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

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

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

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

Only certain items are synced

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

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

Secure iCloud Keychain recovery

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

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

Use of secondary authentication

There are several ways to establish a strong passcode:

- If two-factor authentication is enabled for the user's account, the device passcode is used to recover an escrowed keychain.

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

Keychain escrow process

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

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

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

Escrow security for iCloud Keychain

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

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

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

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

Apple Pay

Apple Pay security overview

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

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

Apple Pay component security

Secure Element

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

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

NFC controller

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

Apple Wallet

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

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

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

Secure Enclave

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

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

Apple Pay servers

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

How Apple Pay uses the Secure Element and NFC controller

Secure Element

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

NFC controller

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

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

Credit, debit, and prepaid cards

Card provisioning security overview

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

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

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

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

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

Adding credit or debit cards to Apple Pay

Adding credit or debit cards manually

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

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

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

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

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

Adding credit or debit cards from an iTunes Store account

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

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

When an 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. This is designed to prevent the information from being read by the Apple device. The provisioning flow is similar to that used for manually added cards, described previously, except one-time passwords are used in lieu of the CVV.

Adding additional verification

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

Payment authorization with Apple Pay

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

Using a shared pairing key

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

Authorizing a secure transaction

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

Using a payment cryptogram for dynamic security

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

- A Terminal Unpredictable Number, for near-field-communication (NFC) transactions

- An Apple Pay server nonce, for transactions within apps

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

Paying with cards using Apple Pay

Paying with cards in stores

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

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

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

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

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

Paying with cards within apps

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

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

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

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

App payment authorization

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

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

Paying with cards at websites

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

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

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

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

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

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

Contactless passes in Apple Pay

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

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

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

Rendering cards unusable with Apple Pay

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

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

Suspending, removing, and erasing cards

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

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

Apple Cash security in iOS, iPadOS, and watchOS

Overview

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

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

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

Using Apple Cash in iMessage

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

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

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

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

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

If the transaction involves:

- A debit card for adding money to Apple Cash

- Providing supplemental money if the Apple Cash balance is insufficient

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

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

Apple Card security

Apple Card application in the Wallet app

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

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

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

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

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

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

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

Apple Card payments and Apple Wallet pass details

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

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

Adding transit and student ID cards to Wallet

Transit cards

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

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

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

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

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

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

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

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

- Erasing the device remotely with Find My

- Enabling Lost Mode with Find My

- Entering a mobile device management (MDM) remote wipe command

- Removing all cards from their Apple ID account page

- Removing all cards from iCloud.com

- Removing all cards from the Wallet app

- Removing the card in the issuer's app

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

Credit and debit cards

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

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

Apple Card and Apple Cash are eligible for Express Transit.

Student ID cards

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

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

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

Student ID cards can be disabled or removed by:

- Erasing the device remotely with Find My

- Enabling Lost Mode with Find My

- Receiving a mobile device management (MDM) remote wipe command

- Removing all cards from their Apple ID account page

- Removing all cards from iCloud.com

- Removing all cards from the Wallet app

- Removing the card in the issuer's app

iMessage

iMessage security overview

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

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

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

How iMessage sends and receives messages securely

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

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

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

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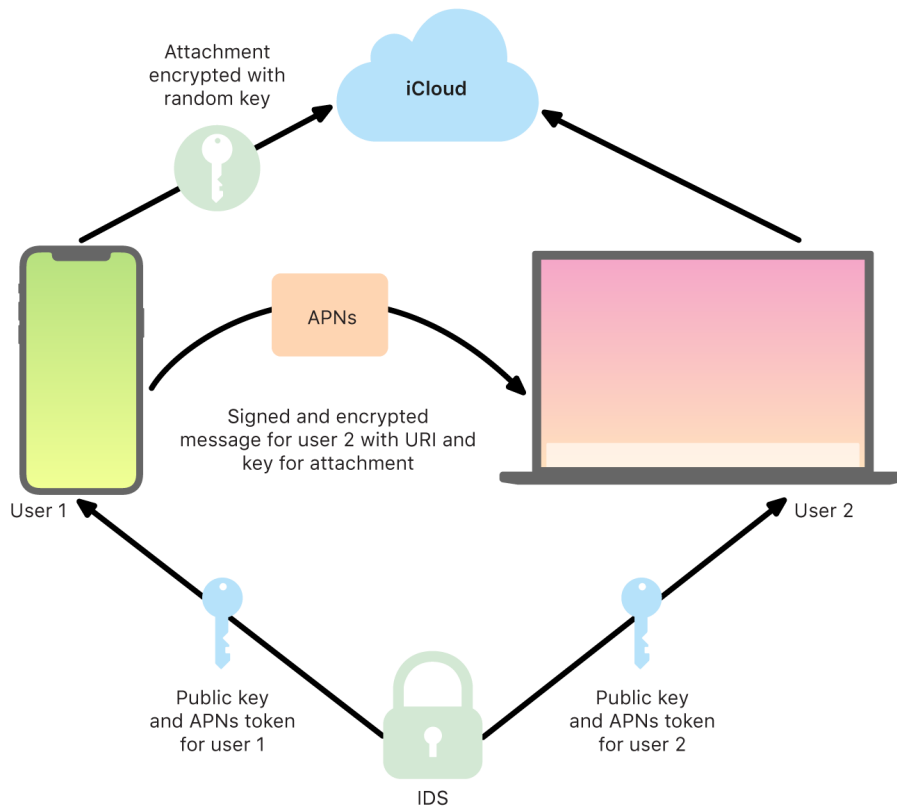
How iMessage sends and receives messages.

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

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

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

Secure iMessage name and photo sharing

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

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

- Name

- Photo

- Photo filename

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

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

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

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

Secure Business Chat using the Messages app

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

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

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

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

The Business Chat service never stores conversation history.

FaceTime security

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

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

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

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

Find My

Find My security

Overview

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

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

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

End-to-end encryption

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

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

When a device goes missing and can't connect to Wi-Fi or cellular—for example, a MacBook Pro is 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 is designed to prevent the various public keys P_i from being linked to the same device.

Keeping users and devices anonymous

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

Using Find My to locate missing Apple devices

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

EC P-224
derived
public key

EC P-224 FindMy
public key
+ Secret
+ Counter

Nearby device
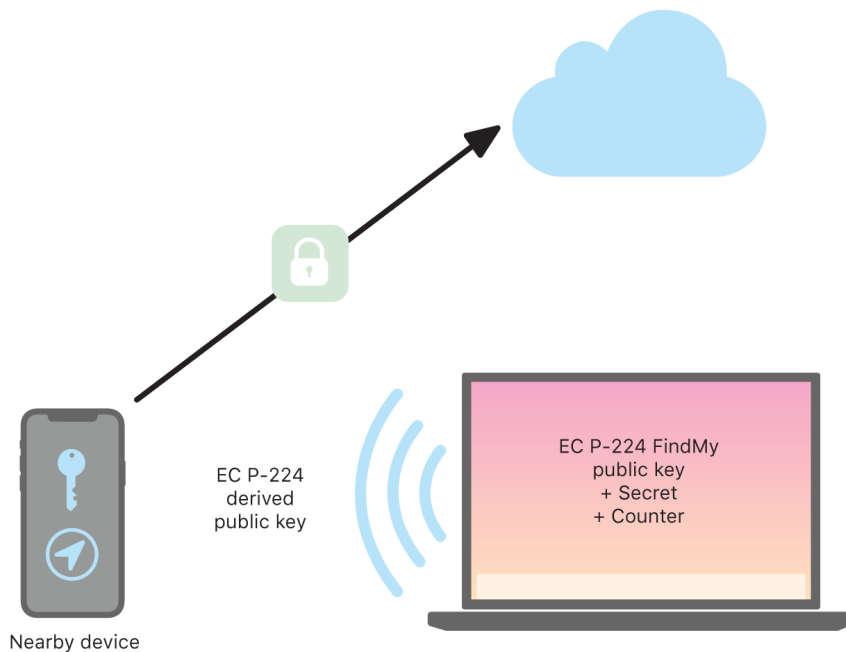
How Find My locates devices.

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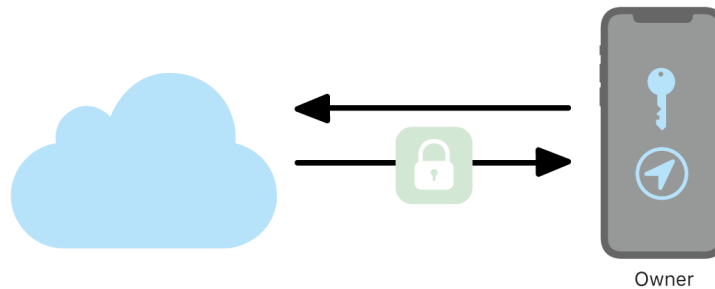
Owner

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

Locating devices that are offline

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

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

Continuity

Continuity security overview

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

Handoff security

Overview

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

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

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

Handoff between native apps and websites

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

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

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

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

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

Handoff of larger data

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

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

Universal Clipboard

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

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

iPhone cellular call relay security

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

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

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

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

iPhone Text Message Forwarding security

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

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

Instant Hotspot security

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

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

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

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

Car keys security in iOS

Overview

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

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

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

Owner pairing

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

Key sharing

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

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

Key deletion

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

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

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

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

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

Standard transactions

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

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

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

Fast transactions

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

Privacy

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

Network security

Network security overview

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

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

TLS security

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

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

App Transport Security

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

- ECDHE_ECDSA_AES and ECDHE_RSA_AES in Galois/Counter Mode (GCM)

- Cipher Block Chaining (CBC) mode

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

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

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

Certificate validity checking

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

IPv6 security

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

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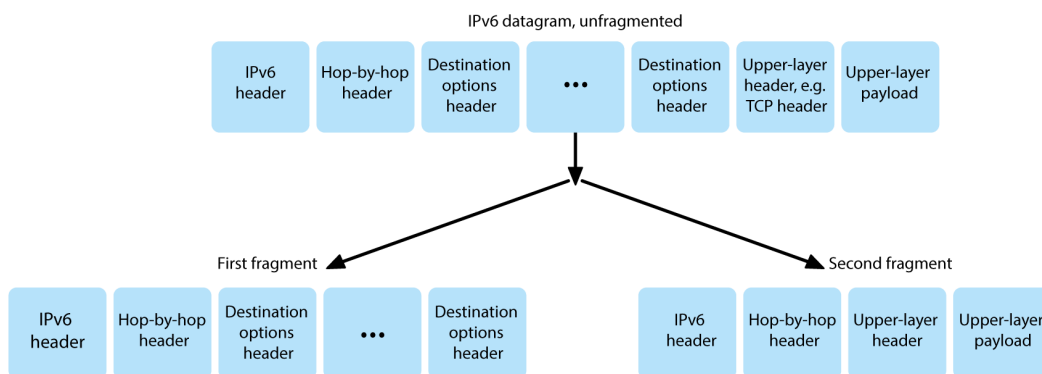
An IPv6 datagram.

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

Virtual private network (VPN) security

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

Protocols supported

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

- IKEv2/IPsec with authentication by shared secret, RSA Certificates, Elliptic Curve Digital Signature Algorithm (ECDSA) Certificates, EAP-MSCHAPv2, or EAP-TLS

- SSL-VPN using the appropriate client app from the App Store

- L2TP/IPsec with user authentication by MS-CHAPV2 password and machine authentication by shared secret (iOS, iPadOS, and macOS) and RSA SecurID or CRYPTOCard (macOS only)

- Cisco IPsec with user authentication by password, RSA SecurID or CRYPTOCard, and machine authentication by shared secret and certificates (macOS only)

VPN deployments supported

iOS, iPadOS, and macOS support the following:

- *VPN On Demand:* For networks that use certificate-based authentication. IT policies specify which domains require a VPN connection by using a VPN configuration profile.

- *Per App VPN:* For facilitating VPN connections on a much more granular basis. Mobile device management (MDM) solutions can specify a connection for each managed app and specific domains in Safari. This helps ensure that secure data always goes to and from the corporate network—and that a user's personal data doesn't.

iOS and iPadOS support the following:

- *Always On VPN:* Can be configured for devices managed through an MDM solution and supervised using Apple Configurator 2, Apple School Manager, or Apple Business Manager. Always On VPN eliminates the need for users to turn on VPN to enable protection when connecting to cellular and Wi-Fi networks. Always On VPN gives an organization full control over device traffic by tunneling all IP traffic back to the organization. The default exchange of parameters and keys for the subsequent encryption, IKEv2 secures traffic transmission with data encryption. The organization can monitor and filter traffic to and from its devices, secure data within its network, and restrict device access to the internet.

Wi-Fi security

Protocol security

Secure access to wireless networks

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

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

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

WPA3 support

WPA3 is supported on the following Apple devices:

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

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

PMF support

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

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

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

Platform protections

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

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

Deprecated protocols

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

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

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

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

Wi-Fi privacy

MAC address randomization

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

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

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

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

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

Wi-Fi frame sequence number randomization

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

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

- iPhone 7 or later
- iPad 5th generation or later

- Apple TV 4K or later

- Apple Watch series 3 or later

- iMac Pro (Retina 5K, 27-inch, 2017) or later

- MacBook Pro (13-inch, 2018) or later

- MacBook Pro (15-inch, 2018) or later

- MacBook Air (Retina, 13-inch, 2018) or later

- Mac mini (2018) or later

- iMac (Retina 4K, 21.5-inch, 2019) or later

- iMac (Retina 5K, 27-inch, 2019) or later

- Mac Pro (2019) or later

Wi-Fi connections

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

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

Hidden networks

Wi-Fi networks are identified by their network name, known as a *service set identifier (SSID)*. Some Wi-Fi networks are configured to hide their SSID, which results in the wireless access point not broadcasting the network's name. These are known as *hidden networks*. iPhone 6s and later devices automatically detect 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 helps prevent the device from broadcasting the name of previously hidden networks a user was connected to, thereby further ensuring privacy.

Bluetooth security

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

- *Pairing:* The process for creating one or more shared secret keys

- *Bonding:* The act of storing the keys created during pairing for use in subsequent connections to form a trusted device pair

- *Authentication:* Verifying that the two devices have the same keys

- *Encryption:* Message confidentiality

- *Message integrity:* Protection against message forgeries

- *Secure Simple Pairing:* Protection against passive eavesdropping and protection against man-in-the-middle attacks

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

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

Support	Bluetooth Classic	Bluetooth Low Energy
Pairing	P-256 elliptic curve	FIPS-approved algorithms (AES-CMAC and P-256 elliptic curve)
Bonding	Pairing information stored in a secure location in iOS, iPadOS, macOS, tvOS, and watchOS devices	Pairing information stored in a secure location in iOS, iPadOS, macOS, tvOS, and watchOS devices
Authentication	FIPS-approved algorithms (HMAC-SHA256 and AES-CTR)	FIPS-approved algorithms
Encryption	AES-CCM cryptography, performed in the Controller	AES-CCM cryptography, performed in the Controller
Message integrity	AES-CCM, used for message integrity	AES-CCM, used for message integrity
Secure Simple Pairing: Protection against passive eavesdropping	Elliptic Curve Diffie-Hellman Exchange (ECDHE)	Elliptic Curve Diffie-Hellman Exchange (ECDHE)
Secure Simple Pairing: Protection against man-in-the-middle (MITM) attacks	Two user-assisted numeric methods: numerical comparison or passkey entry	Two user-assisted numeric methods: numerical comparison or passkey entry Pairings require a user response, including all non-MITM pairing modes
Bluetooth 4.1 or later	iMac Late 2015 or later MacBook Pro Early 2015 or later	iOS 9 or later iPadOS 13.1 or later macOS 10.12 or later tvOS 9 or later watchOS 2.0 or later
Bluetooth 4.2 or later	iPhone 6 or later	iOS 9 or later iPadOS 13.1 or later macOS 10.12 or later tvOS 9 or later watchOS 2.0 or later

Bluetooth Low Energy privacy

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

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

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

Ultra Wideband security in iOS

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

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

Single sign-on security

Single sign-on

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

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

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

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

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

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

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

Extensible Single sign-on

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

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

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

AirDrop security

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

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

AirDrop operation

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

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

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

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

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

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

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

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

Wi-Fi password sharing security on iPhone and iPad

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

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

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

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

Firewall security in macOS

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

- Block all incoming connections, regardless of app.

- Automatically allow built-in software to receive incoming connections.

- Automatically allow downloaded and signed software to receive incoming connections.

- Add or deny access based on user-specified apps.

- Prevent the Mac from responding to ICMP (Internet Control Message Protocol) probing and portscan requests.

Developer kit security

Developer kit security overview

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

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

HomeKit

HomeKit communication security

Overview

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

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

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

Communication between HomeKit accessories

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

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

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

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

HomeKit and Siri

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

HomeKit data security

HomeKit data updates between devices and users

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

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

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

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

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

Home data and apps

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

Local data storage

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

Securing routers with HomeKit

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

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

- *No restriction:* Allow unrestricted access to the internet and the local network.

- *Automatic:* This is the default setting. Allow access to the internet and the local network based on a list of internet sites and local ports provided to Apple by the accessory manufacturer. This list includes all sites and ports needed by the accessory in order to function properly. (No Restriction is in place until such a list is available.)

- *Restrict to Home:* No access to the internet or the local network except for the connections required by HomeKit to discover and control the accessory from the local network (including from the home hub to support remote control).

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

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

HomeKit camera security

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

HomeKit secure video

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

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

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

HomeKit security with Apple TV

Using third-party remote accessories with Apple TV

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

Apple TV profiles for HomeKit homes

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

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

HomeKit accessories and iCloud

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

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

Mutual authentication of an accessory and an Apple device

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

Accessory setup process

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

Accessory list of allowed users

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

How accessories connect to iCloud remote access server

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

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

CloudKit security

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

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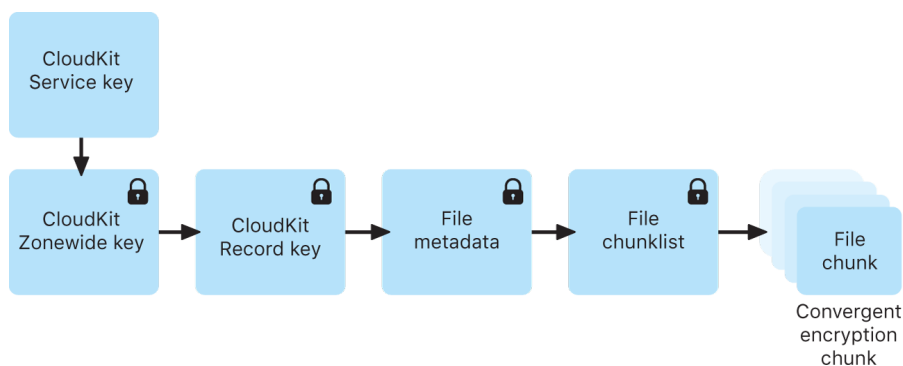
CloudKit end-to-end encryption.

SiriKit security for iOS, iPadOS, and watchOS

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

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

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

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

DriverKit security for macOS 10.15

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

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

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

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

ReplayKit security in iOS and iPadOS

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

Movie recording

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

- *Permissions dialog:* Before recording starts, ReplayKit presents a user consent alert requesting that the user acknowledge their intent to record the screen, the microphone, and the front-facing camera. This alert is presented once per app process, and it's presented again if the app is left in the background for longer than 8 minutes.

- *Screen and audio capture:* Screen and audio capture occurs out of the app's process in the ReplayKit daemon replayd. This is designed to ensure 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 the ReplayKit subsystems and is never accessible to any apps. This helps prevent recordings being used by third parties without the user's consent.

- *End-user preview and sharing:* The user has the ability to preview and share the movie with a user interface vended by ReplayKit. The user interface is presented out-of-process through the iOS Extension infrastructure and has access to the generated movie file.

ReplayKit broadcasting

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

- *Screen and audio capture:* The screen and audio capture mechanism during broadcasting is identical to movie recording and occurs in `replayd`.

- *Broadcast extensions:* For third-party services to participate in ReplayKit broadcasting, they're required to create two new extensions that are configured with the com.apple.broadcast-services endpoint:

  - A user interface extension that allows the user to set up their broadcast

  - An upload extension that handles uploading video and audio data to the service's back-end servers

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

- *Broadcast picker:* With the broadcast picker, users initiate system broadcasts directly from their app using the same system-defined user interface that's accessible using Control Center. The user interface is implemented using a private API and is an extension that lives within the ReplayKit framework. It is out-of-process from the hosting app.

- *Upload extension:* The extension that third-party broadcast services implement to handle video and audio content during broadcasting uses raw unencoded sample buffers. During this mode of handling, video and audio data is serialized and passed to the third-party upload extension in real time through a direct XPC connection. Video data is encoded by extracting the IOSurface object from the video sample buffer, encoding it securely as an XPC object, sending it over through XPC to the third-party extension, and decoding it securely back into an IOSurface object.

ARKit security in iOS and iPadOS

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

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

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

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

Secure device management

Secure device management overview

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

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

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

Pairing model security for iPhone and iPad

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

- That require pairing can't be started until after the device has been unlocked by the user

- Won't start unless the device has been recently unlocked

- May (such as with photo syncing) require the device to be unlocked to begin

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

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

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

Mobile device management

Mobile device management security overview

Overview

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

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

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

Enrollment types

- *User Enrollment:* User Enrollment is designed for devices owned by the user and is integrated with Managed Apple IDs to establish a user identity on the device. Managed Apple IDs are part of the User Enrollment profile, and the user must successfully authenticate in order for enrollment to be completed. Managed Apple IDs can be used alongside a personal Apple ID that the user has already signed in with. Managed apps and accounts use a Managed Apple ID, and personal apps and accounts use a personal Apple ID.

- *Device Enrollment:* Device Enrollment allows organizations to have users manually enroll devices and then manage many different aspects of device use, including the ability to erase the device. Device Enrollment also has a larger set of payloads and restrictions that can be applied to the device. When a user removes an enrollment profile, all configuration profiles, their settings, and managed apps based on that enrollment profile are removed with it.

- *Automated Device Enrollment:* Automated Device Enrollment lets organizations configure and manage devices from the moment the devices are removed from the box (in a process known as *Auto Advance deployment*). These devices are known as *supervised*, and users have the option to prevent the MDM profile from being removed by the user. Automated Device Enrollment is designed for devices owned by the organization.

Device restrictions

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

Passcode and password settings management

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

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

Configuration profile enforcement

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

Configuration profiles

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

Enrollment profiles

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

Configuration profile settings

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

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

Profile signing and encryption

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

Profile installation

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

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

Automated Device Enrollment

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

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

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

The setup process for users can be further simplified by removing specific steps in the Setup Assistant for devices so that so users are up and running quickly. Administrators can also control whether users can remove the MDM profile from the device and help ensure that device restrictions are in place throughout the life cycle of the device. After the device is unboxed and activated, it can enroll in the organization's MDM solution—and all management settings, apps, and books are installed as defined by the MDM administrator.

Apple School Manager and Apple Business Manager

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

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

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

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

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

Device supervision

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

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

- Using Apple Configurator 2 to supervise the device

  During this process, the device is erased and all data is lost.

- Enrolling the device in an MDM solution and selecting supervision as part of the enrollment process

Mac computers can be supervised if they:

- Are running macOS 11 enrolled in MDM using device enrollment

- Are upgraded to macOS 11 and the enrollment in MDM *was* a user approved MDM enrollment

- Are running macOS 10.14.4 or later and:

  - The devices' serial numbers appear in Apple School Manager or Apple Business Manager

  - Are enrolled in an MDM solution using Apple School Manager or Apple Business Manager

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

- iPhone and iPod touch with iOS 13 or later

- iPad with iPadOS 13.1 or later

- Apple TV with tvOS 13 or later

- Mac computers with macOS 10.14.4 or later

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

Activation Lock security

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

Behavior on iPhone and iPad

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

Behavior on a Mac with Apple silicon

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

- There is no LocalPolicy for the current macOS

- The LocalPolicy is invalid for that macOS

- The LocalPolicy nonce hash values don't match the hashes of values stored in the Secure Storage Component

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

Behavior on Intel-based Mac computers

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

Managed Lost Mode and remote wipe

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

Managed Lost Mode

If a supervised iOS or iPadOS device with iOS 9 or later is lost or stolen, a mobile device management (MDM) administrator can remotely enable Lost Mode (called Managed Lost Mode) on that device. When Managed 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. The administrator can also request the device to send its current location (even if Location Services are off) and, optionally, play a sound. When an administrator turns off Managed 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

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. For remote wipe through Microsoft Exchange ActiveSync, the device checks in with the Microsoft Exchange Server before performing the wipe.

When a remote wipe command is triggered by MDM or iCloud, the iPhone, iPad, iPod touch, or Mac device sends an acknowledgment back to the MDM solution and performs the wipe.

Remote wipe isn't possible in the following situations:

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

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

Shared iPad security in iPadOS

Overview

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

Signing in to Shared iPad

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

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

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

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

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

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

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

Signing out of Shared iPad

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

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

Apple Configurator 2 security

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

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

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

Screen Time security

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

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

The table below describes the main features of Screen Time.

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

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

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

Parents and Screen Time

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

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

Screen Time analytics

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

- Was Screen Time turned on during Setup Assistant or later in Settings

- Change in Category usage after creating a limit for it (within 90 days)

- Is Screen Time turned on

- Is Downtime enabled

- Number of times the "Ask for more" query was used

- Number of app limits

- Number of times users viewed usage in the Screen Time settings, per user type and per view type (local, remote, widget)

- Number of times users ignore a limit, per user type

- Number of times users delete a limit, per user type

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

Glossary

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

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

AES cryptographic engine A dedicated hardware component that implements AES.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

NAND Nonvolatile flash memory.

Passcode-derived key (PDK) The encryption key derived from the entangling of the user password with the long-term SKP key and the UID of the Secure Enclave.

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

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

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

Sealed Key Protection (SKP) A technology in Data Protection that protects, or *seals*, encryption keys with measurements of system software and keys available only in hardware (such as the UID of the Secure Enclave).

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Document revision history

Date	Summary
May, 2021	Updated for: • iOS 14.5 • iPadOS 14.5 • macOS 11.3 • tvOS 14.5 • watchOS 7.4 Topics added: • Magic Keyboard with Touch ID. • Secure intent and connections to the Secure Enclave. • Auto Unlock and Apple Watch. • CustomOS Image4 Manifest Hash (coih). Topics edited: • Added two new Express Mode transactions in Express Cards with power reserve. • Edited Secure Enclave feature summary. • Software update content added to Secure Multi-Boot (smb3). • Additional content for Sealed Key Protection (SKP).

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

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

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

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