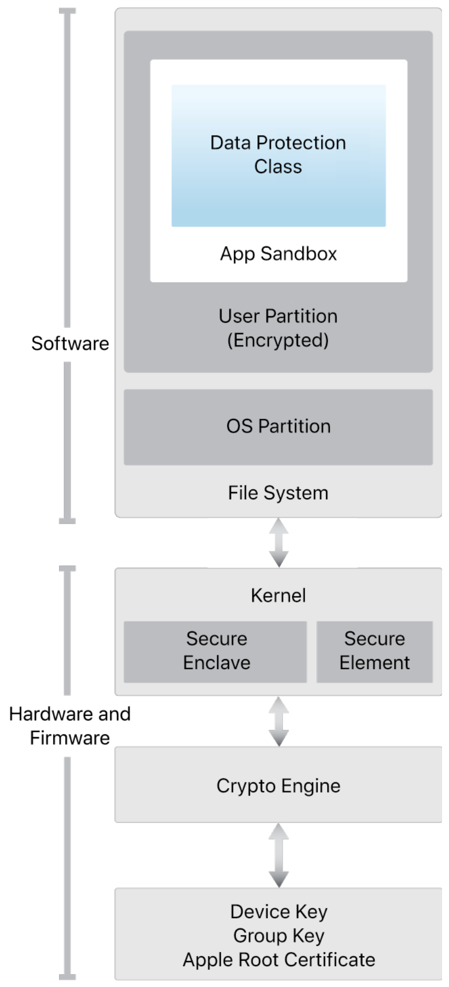
iOS Security

iOS 12

September 2018

Contents

Introduction



Security architecture diagram of iOS provides a visual overview of the different technologies discussed in this document.

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

Every iOS device combines software, hardware, and services designed to work together for maximum security and a transparent user experience. iOS protects not only the device and its data at rest, but the entire ecosystem, including everything users do locally, on networks, and with key internet services.

iOS and iOS devices provide advanced security features, and yet they're also easy to use. Many of these features are enabled by default, so IT departments don't need to perform extensive configurations. And key security features like device encryption aren't configurable, so users are unable to disable them by mistake. Other features, such as Face ID, enhance the user experience by making it simpler and more intuitive to secure the device.

This document provides details about how security technology and features are implemented within the iOS platform. It will also help organizations combine iOS platform security technology and features with their own policies and procedures to meet their specific security needs.

This document is organized into the following topic areas:

- **System security:** The integrated and secure software and hardware that are the platform for iPhone, iPad, and iPod touch.
- **Encryption and data protection:** The architecture and design that protects user data if the device is lost or stolen, or if an unauthorized person attempts to use or modify it.
- **App security:** The systems that enable apps to run securely and without compromising platform integrity.
- **Network security:** Industry-standard networking protocols that provide secure authentication and encryption of data in transmission.
- **Apple Pay:** Apple's implementation of secure payments.
- **Internet services:** Apple's network-based infrastructure for messaging, syncing, and backup.
- **User password management:** Password restrictions and access to passwords from other authorized sources.
- **Device controls:** Methods that allow management of iOS devices, prevent unauthorized use, and enable remote wipe if a device is lost or stolen.
- **Privacy controls:** Capabilities of iOS that can be used to control access to Location Services and user data.
- **Security Certifications and programs:** Information on ISO certifications, Cryptographic validation, Common Criteria Certification, and commercial solutions for classified (CSfC).

System Security

System security is designed so that both software and hardware are secure across all core components of every iOS device. This includes the boot-up process, software updates, and Secure Enclave. This architecture is central to security in iOS, and never gets in the way of device usability.

The tight integration of hardware, software, and services on iOS devices ensures that each component of the system is trusted, and validates the system as a whole. From initial boot-up to iOS software updates to third-party apps, each step is analyzed and vetted to help ensure that the hardware and software are performing optimally together and using resources properly.

Secure boot chain

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

When an iOS device is turned on, its application processor immediately executes code from read-only memory known as **Boot ROM**. This immutable code, known as the hardware root of trust, is laid down during chip fabrication, and is implicitly trusted. The Boot ROM code contains the Apple Root CA public key, which is 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 where each step ensures that the next is signed by Apple. When the iBoot finishes its tasks, it verifies and runs the iOS 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 of the Boot ROM to load LLB (on older devices) or iBoot (on newer devices) results in the device entering DFU mode. In the case of a failure in LLB or iBoot to load or verify the next step, startup is halted and the device displays the connect to iTunes screen. This is known as recovery mode. In either case, the device must be connected to iTunes 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:

- **Recovery Mode:** Set by iBoot on devices with Apple A10, S2, and newer **system on chip (SoCs)**
- **DFU Mode:** Set by Boot ROM on devices with an A12 SoC

For more information, see the "Encryption and Data Protection" section of this paper.

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

Entering Device Firmware Upgrade (DFU) mode

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.

First connect the device to a computer using a USB cable.

Then, based on device, do the following:

iPhone X, iPhone 8, or iPhone 8 Plus. Press and quickly release the Volume Up button. Press and quickly release the Volume Down button. Then, press and hold the side button until you see the recovery mode screen.

iPhone 7 or iPhone 7 Plus. Press and hold the side and Volume Down buttons at the same time. Keep holding them until you see the recovery mode screen.

iPhone 6s and earlier, iPad, or iPod touch. Press and hold both the Home and the Top (or side) buttons at the same time. Keep holding them until you see the recovery mode screen.

Note: Nothing will be displayed on the screen when the device is in DFU mode. If the Apple logo appears, the side or Sleep/Wake button was held down too long.

The Secure Enclave coprocessor also utilizes a secure boot process that ensures its separate software is verified and signed by Apple. See the "Secure Enclave" section of this paper.

For more information on manually entering recovery mode, go to: https://support.apple.com/kb/HT1808

System Software Authorization

Apple regularly releases software updates to address emerging security concerns and also provide new features; these updates are provided for all supported devices simultaneously. Users receive iOS update notifications on the device and through iTunes, and updates are delivered wirelessly, encouraging rapid adoption of the latest security fixes.

The startup process described previously helps ensure that only Apple-signed code can be installed on a device. To prevent devices from being downgraded to older versions that lack the latest security updates, iOS uses a process called *System Software Authorization*. If downgrades were possible, an attacker who gains possession of a device could install an older version of iOS and exploit a vulnerability that's been fixed in the newer version.

On a device with Secure Enclave, the Secure Enclave coprocessor also utilizes System Software Authorization to ensure the integrity of its software and prevent downgrade installations. See the "Secure Enclave" section of this paper.

iOS software updates can be installed using iTunes or over the air (OTA) on the device. With iTunes, a full copy of iOS is downloaded and installed. OTA software updates download only the components required to complete an update, improving network efficiency, rather than downloading the entire OS. Additionally, software updates can be cached on a Mac running macOS High Sierra with Content Caching turned on, so that iOS devices don't need to redownload the necessary update over the Internet. They'll still need to contact Apple servers to complete the update process.

During an iOS upgrade, iTunes (or the device itself, in the case of OTA software updates) connects to the Apple installation authorization server and sends it a list of cryptographic measurements for each part of the installation bundle to be installed (for example, iBoot, the kernel, and OS image), a random anti-replay value (nonce), and the device's unique **Exclusive Chip Identification (ECID)**.

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

The boot-time chain-of-trust evaluation verifies that the signature comes from Apple and that the measurement of the item loaded from disk, combined with the device's ECID, matches what was covered by the signature.

These steps ensure that the authorization is for a specific device and that an old iOS version from one device can't be copied to another. The nonce prevents an attacker from saving the server's response and using it to tamper with a device or otherwise alter the system software.

Secure Enclave

The Secure Enclave is a coprocessor fabricated within the system on chip (SoC). It uses encrypted memory and includes a hardware random number generator. The Secure Enclave provides all cryptographic operations for **Data Protection** key management and maintains the integrity of Data Protection even if the kernel has been compromised. Communication between the Secure Enclave and the application processor is isolated to an interrupt-driven mailbox and shared memory data buffers.

The Secure Enclave includes a dedicated Secure Enclave Boot ROM. Similar to the application processor Boot ROM, the Secure Enclave Boot ROM is immutable code that establishes the hardware root of trust for the Secure Enclave.

The Secure Enclave runs a Secure Enclave OS based on an Apple-customized version of the L4 microkernel. This Secure Enclave OS is signed by Apple, verified by the Secure Enclave Boot ROM, and updated through a personalized software update process.

When the device starts up, an ephemeral memory protection key is created by the Secure Enclave Boot ROM, entangled with the device's UID, and used to encrypt the Secure Enclave's portion of the device's memory space. Except on the Apple A7, the Secure Enclave memory is also authenticated with the memory protection key. On A11 and newer and S3 and newer SOCs, an integrity tree is used to prevent replay of security-critical Secure Enclave memory, authenticated by the memory protection key and nonces stored in on-chip SRAM.

Data saved to the file system by the Secure Enclave is encrypted with a key entangled with the UID and an anti-replay counter. The anti-replay counter is stored in a dedicated nonvolatile memory **integrated circuit (IC)**.

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

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
- Touch ID or Face ID enable/disable
- Touch ID fingerprint add/delete
- Face ID reset
- Apple Pay card add/remove
- Erase All Content and Settings

The Secure Enclave is also responsible for processing fingerprint and face data from the Touch ID and Face ID sensors, determining if there's a match, and then enabling access or purchases on behalf of the user.

OS Integrity Protection

Kernel Integrity Protection

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

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

System Coprocessor Integrity Protection

System coprocessors are CPUs on the same SoC as the application processor. System coprocessors are dedicated to a specific purpose, and the iOS kernel delegates many tasks to them. Examples include:

- Secure Enclave
- Image Sensor Processor
- Motion coprocessor

Because coprocessor firmware handles many critical system tasks, its security is a key part of the overall system's security.

System Coprocessor Integrity Protection (SCIP) uses a mechanism similar to Kernel Integrity Protection to prevent modification of coprocessor firmware. At boot time, iBoot loads each coprocessor's firmware into a protected memory region, reserved and separate from the KIP region. iBoot configures each coprocessor's memory management units to prevent:

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

The Secure Enclave Operating System is responsible for configuring the Secure Enclave's SCIP at boot time.

The hardware used to enable SCIP is locked after the boot process completes to prevent reconfiguration. SCIP is supported on SoCs starting with the A12 and S4.

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 prevent modification of function pointers and return addresses (code pointers). Doing so increases the difficulty of many attacks. For example, a Return Oriented Programming (ROP) attack attempts to trick the device into executing existing code maliciously by manipulating function return addresses stored on the stack.

PAC is supported on A12 and S4 SoCs.

Touch ID

Touch ID is the fingerprint sensing system that makes secure access to iPhone and iPad 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.

Face ID

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

Touch ID, Face ID, and passcodes

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

You can use your passcode anytime instead of Touch ID or Face ID, but the following operations always require a passcode instead of a biometric:

- Updating your software.
- Erasing your device.
- Viewing or changing passcode settings.
- Installing iOS configuration profiles.

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

- The device has just been turned on or restarted.
- The device hasn't been unlocked for more than 48 hours.
- The passcode hasn't been used to unlock the device in the last 156 hours (six and a half days) and a biometric hasn't unlocked the device in the last 4 hours.
- The device has received a remote lock command.
- After five unsuccessful biometric match attempts.
- After initiating power off/Emergency SOS.

When Touch ID or Face ID is enabled, the device immediately locks when the side 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 your iPhone 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 is required to obtain access to your device. With Face ID, the probability of a false match is different for twins and siblings who look like you and for children under the age of 13 (because their distinct facial features may not have fully developed). If you're concerned about this, Apple recommends using a passcode to authenticate.

Touch ID security

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

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

Face ID security

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 your face when you wake Apple devices that feature Face ID by raising it or tapping the screen, as well as when those devices attempt to authenticate you 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 your eyes are open and your attention is directed at your device; for accessibility, this is disabled when VoiceOver is activated and, if required, can be disabled separately.

After it confirms the presence of an attentive face, the TrueDepth camera projects and reads over 30,000 infrared dots to form a depth map of the face, along with a 2D infrared image. This data is used to create a sequence of 2D images and depth maps, which are digitally signed and sent to the Secure Enclave. To counter both digital and physical spoofs, the TrueDepth camera randomizes the sequence of 2D images and depth map captures, and projects a device-specific random pattern. A portion of the A11 and newer SoCs neural engine—protected within the Secure Enclave—transforms this data into a mathematical representation and compares that representation to the enrolled facial data. This enrolled facial data is itself a mathematical representation of your face captured across a variety of poses.

Facial matching is performed within the Secure Enclave using neural networks trained specifically for that purpose. We developed the facial matching neural networks using over a billion images, including IR and depth images collected in studies conducted with the participants' informed consent. Apple 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, glasses, contact lenses, and many sunglasses. Furthermore, it's designed to work indoors, outdoors, and even in total darkness. An additional neural network that's trained to spot and resist spoofing defends against attempts to unlock your iPhone X with photos or masks.

Face ID data, including mathematical representations of your face, is encrypted and available only to the Secure Enclave. This data never leaves the device. It isn't 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 your face calculated during enrollment.
- The mathematical representations of your 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.

How Touch ID or Face ID unlocks an iOS device

With Touch ID or Face ID disabled, when a device 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 you unlock the device by entering your passcode.

With Touch ID or Face ID enabled, the keys aren't discarded when the device locks; instead, they're wrapped with a key that's given to the Touch ID or Face ID subsystem inside the Secure Enclave. When you attempt to unlock the device, if the device detects a successful match, it provides the key for unwrapping the Data Protection keys, and the device 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 are lost; they're discarded by the Secure Enclave after any conditions are met that require passcode entry (for example, after not being unlocked for 48 hours or after five failed match attempts).

To improve unlock performance and keep pace with the natural changes of your face and look, Face ID augments its stored mathematical representation over time. Upon successful unlock, Face ID may use the newly calculated mathematical representation—if its quality is sufficient—for a finite number of additional unlocks before that data is discarded. Conversely, if Face ID fails to recognize you, but the match quality is higher than a certain threshold and you immediately follow the failure by entering your 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 you stop matching against it and after a finite number of unlocks. These augmentation processes allow Face ID to keep up with dramatic changes in your facial hair or makeup use, while minimizing false acceptance.

Touch ID, Face ID, and Apple Pay

You can also use Touch ID and Face ID with Apple Pay to make easy and secure purchases in stores, apps, and on the web. For more information on Touch ID and Apple Pay, see the Apple Pay section of this paper.

To authorize an in-store payment with Face ID, you must first confirm intent to pay by double-clicking the side button. You then authenticate using Face ID before placing your iPhone X near the contactless payment reader. If you'd like to select a different Apple Pay payment method after Face ID authentication, you'll need to reauthenticate, but you won't have to double-click the side button again.

To make a payment within apps and on the web, you confirm intent to pay by double-clicking the side button, then authenticate using Face ID to authorize the payment. If your Apple Pay transaction isn't completed within 30 seconds of double-clicking the side button, you'll have to reconfirm intent to pay by double-clicking again.

Face ID Diagnostics

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

As part of setting up Face ID Diagnostics, your existing Face ID enrollment will be deleted and you'll be asked to re-enroll in Face ID. On devices that support Face ID, they will begin recording Face ID images captured during authentication attempts for the next 10 days; they will automatically stop saving images thereafter. Face ID Diagnostics doesn't automatically send data to Apple. You can review and approve enrollment and unlock images (both successful and failed) included in Face ID diagnostic data that's gathered while in diagnostics mode before it's sent to Apple. Face ID Diagnostics will upload only the Face ID Diagnostics images you have approved. The data is encrypted before it's uploaded and is immediately deleted from the device after the upload completes. Images you reject are immediately deleted.

If you don't conclude the Face ID Diagnostics session by reviewing images and uploading any approved images, Face ID Diagnostics will automatically end after 40 days and all diagnostic images will be deleted from the device. You can also disable Face ID Diagnostics at any time. All local images are immediately deleted if you do so, and no Face ID data is shared with Apple in these cases.

Other uses for Touch ID and Face ID

Third-party apps can use system-provided APIs to ask the user to authenticate using Touch ID or Face ID or a passcode, 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. Keychain items can also be protected with Touch ID or Face ID, to be released by the Secure Enclave only by a successful match or the device passcode. App developers have APIs to verify that a passcode has been set by the user, before requiring Touch ID or Face ID or a passcode to unlock Keychain items. App developers can do 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 ECC keys inside Secure Enclave that can be protected by Touch ID or Face ID. Operations with these keys are always performed inside the Secure Enclave after it authorizes their use.

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

Encryption and Data Protection

Erase all content and settings

The "Erase all content and settings" option in Settings obliterates all of the keys in Effaceable Storage, rendering all user data on the device cryptographically inaccessible. Therefore, it's an ideal way to be sure all personal information is removed from a device before giving it to somebody else or returning it for service.

Important: Don't use the "Erase all content and settings" option until device has been backed up, as there is no way to recover the erased data.

The secure boot chain, code signing, and runtime process security all help to ensure that only trusted code and apps can run on a device. iOS has additional encryption and data protection features to safeguard user data, even in cases where other parts of the security infrastructure have been compromised (for example, on a device with unauthorized modifications). This provides important benefits for both users and IT administrators, protecting personal and corporate information at all times and providing methods for instant and complete remote wipe in the case of device theft or loss.

Hardware security features

On mobile devices, speed and power efficiency are critical. Cryptographic operations are complex and can introduce performance or battery life problems if not designed and implemented with these priorities in mind.

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

The device's **unique IDs (UIDs)** and a device **group IDs (GIDs)** are AES-256 bit keys fused (UID) or compiled (GID) into the application processor and Secure Enclave during manufacturing. No software or firmware can read them directly; they can see only the results of encryption or decryption operations performed by dedicated AES engines implemented in silicon using the UID or GID as a key. The application processor and Secure Enclave each have their own UID and GID. The Secure Enclave UID and GID can only be used by the AES engine dedicated to the Secure Enclave. The UIDs and GIDs are also not available through **Joint Test Action Group (JTAG)** or other debugging interfaces.

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

Software running on the Secure Enclave takes advantage of the UID to protect device-specific secrets. The UID allows data to be cryptographically tied to a particular device. For example, the key hierarchy protecting the file system includes the UID, so if the memory chips are physically moved from one device to another, the files are inaccessible. The UID isn't related to any other identifier on the device.

The GID is common to all processors in a class of devices (for example, all devices using the Apple A8 processor).

Apart from the UID and GID, all other cryptographic keys are created by the system's random number generator (RNG) using an algorithm based on

CTR_DRBG source code. System entropy is generated from timing variations during boot, and additionally from interrupt timing after the device has booted. Keys generated inside the Secure Enclave use its true hardware random number generator based on multiple ring oscillators post processed with CTR_DRBG.

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

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

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

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

File Data Protection

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

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

Architecture overview

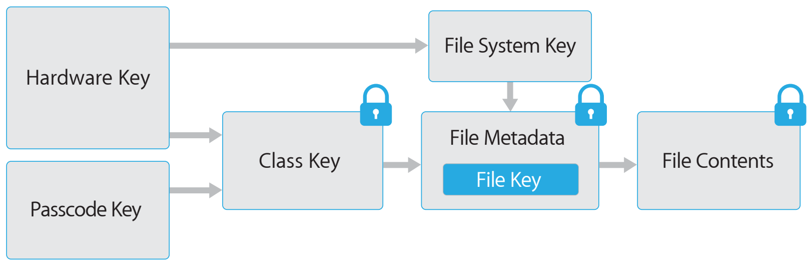
Every time a file on the data partition is created, Data Protection creates a new 256-bit key (the "per-file" key) and gives it to the hardware AES engine, which uses the key to encrypt the file as it is written to flash memory using AES-XTS mode. On devices with an A7, S2, or S3 SoC, AES-CBC is used. The initialization vector is calculated with the block offset into the file, encrypted with the SHA-1 hash of the **per-file key**.

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

Devices running with the Apple File System 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 will be 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, then supplied to the hardware AES engine, which decrypts the file as it is read from flash memory. All wrapped file key handling occurs in the Secure Enclave; the file key is never directly exposed to the application processor. At boot time, the Secure Enclave negotiates an ephemeral key with the AES engine. When the Secure Enclave unwraps a file's keys, they are rewrapped with the ephemeral key and sent back to the application processor.

The metadata of all files in the file system is encrypted with a random key, which is created when iOS is first installed or when the device is wiped by a user. On devices that support the Apple File System, the file system metadata key is wrapped by the Secure Enclave UID key for long-term storage. Just like per-file or per-extent keys, the metadata key is never directly exposed to the application processor; the Secure Enclave provides an ephemeral, per-boot version instead. When stored, the encrypted file system key is additionally wrapped by an "effaceable key" stored in Effaceable Storage. This key doesn't provide additional confidentiality of data. Instead, it's designed to be quickly erased on demand (by the user with the "Erase All Content and Settings" option, or by a user or administrator issuing a remote wipe command from an MDM solution, 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.

Passcodes

Passcode considerations

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.

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

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

Delays between passcode attempts

Attempts	Delay Enforced
1–4	none
5	1 minute
6	5 minutes
7–8	15 minutes
9	1 hour

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

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

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

To improve security while maintaining usability, iOS 11.4.1 or later requires Touch ID, Face ID, or passcode entry to activate the USB interface if USB hasn't been used recently. This eliminates attack surface against physically connected devices such as malicious chargers while still enabling usage of USB accessories within reasonable time constraints. If more than an hour has passed since the iOS device has locked or since a USB connection has been detached, the device won't allow any new connections to be established until the device is unlocked. This hour period:

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

In addition, on iOS 12 if it's been more than three days since a USB connection has been established, the device will disallow new USB connections immediately after it locks. This is to increase protection for users that don't often make use of such connections. USB connections are also disabled whenever the device is in a state where it requires a passcode to re-enable biometric authentication.

The user can choose to re-enable always-on USB connections in Settings, and setting up some assistive devices does this automatically.

DFU and Recovery Mode

On devices with Apple A10, A11, and S3 SoCs, class keys protected by the user’s passcode can’t be accessed from Recovery Mode. TheA12 and S4 SoCs extend this protection to DFU mode.

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

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

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

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

Data Protection Classes

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

Complete Protection

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

Protected Unless Open

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

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

Protected Until First User Authentication

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

No Protection

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

Data Protection class key

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

Keychain data protection

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

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

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 apps can access. Keychain access APIs result in calls to the daemon, which queries the app's "Keychain-access-groups," "application-identifier," and "application-group" entitlements. Rather than limiting access to a single process, access groups allow Keychain items to be shared between apps.

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

Components of a Keychain item

Along with the access group, each Keychain item contains administrative metadata (such as "created" and "last updated" timestamps).

It also contains SHA-1 hashes of the attributes used to query for the item (such as the account and server name) to allow lookup without decrypting each item. And finally, it contains the encryption data, which includes the following:

• Version number

• Access control list (ACL) data

• Value indicating which protection class the item is in

• Per-item key wrapped with the protection class key

• Dictionary of attributes describing the item (as passed to SecItemAdd), encoded as a binary plist and encrypted with the per-item key

The encryption is AES-128 in GCM (Galois/Counter Mode); the access group is included in the attributes and protected by the GMAC tag calculated during encryption.

are enforced through code signing, **Provisioning Profiles**, and the Apple Developer Program.

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

Availability	File Data Protection	Keychain Data Protection
When unlocked	NSFileProtectionComplete	kSecAttrAccessibleWhenUnlocked
While locked	NSFileProtectionCompleteUnlessOpen	N/A
After first unlock	NSFileProtectionCompleteUntilFirstUserAuthentication	kSecAttrAccessibleAfterFirstUnlock
Always	NSFileProtectionNone	kSecAttrAccessibleAlways
Passcode enabled	N/A	kSecAttrAccessible-WhenPasscodeSetThisDeviceOnly

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

The class `kSecAttrAccessibleWhenPasscodeSetThisDeviceOnly` behaves the same as `kSecAttrAccessibleWhenUnlocked`; however, it is 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. For example, a VPN certificate must always be available so the device keeps a continuous connection, but it's classified as "non-migratory," so it can't be moved to another device.

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

Item	Accessible
Wi-Fi passwords	After first unlock
Mail accounts	After first unlock
Exchange accounts	After first unlock
VPN passwords	After first unlock
LDAP, CalDAV, CardDAV	After first unlock
Social network account tokens	After first unlock
Handoff advertisement encryption keys	After first unlock
iCloud token	After first unlock
Home sharing password	When unlocked
Find My iPhone token	Always
Voicemail	Always

iTunes backup	When unlocked, non-migratory
Safari passwords	When unlocked
Safari bookmarks	When unlocked
VPN certificates	Always, non-migratory
Bluetooth® keys	Always, non-migratory
Apple Push Notification service (APNs) token	Always, non-migratory
iCloud certificates and private key	Always, non-migratory
iMessage keys	Always, non-migratory
Certificates and private keys installed by a configuration profile	Always, non-migratory
SIM PIN	Always, non-migratory

Keychain access control

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

Keybags

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

User keybag is where the wrapped class keys used in normal operation of the device are stored. For example, when a passcode is entered, the `NSFileProtectionComplete` key is loaded from the user keybag and unwrapped. It is a binary property list (.plist) file stored in the No Protection class, whose contents are encrypted with a key held in Effaceable Storage. In order to give forward security to keybags, this key is wiped and regenerated each time a user changes their passcode. The `AppleKeyStore` kernel extension 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 is used to store the wrapped class keys used for operations involving device-specific data. iOS 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 doesn't support cryptographic separation of per-user file system content, which means the system will use 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. On iOS 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 is created when an encrypted backup is made by iTunes 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 re-encrypted to these new keys. As explained previously, non-migratory Keychain items remain wrapped with the UID-derived key, allowing them to be restored to the device they were originally backed up from, but rendering them inaccessible on a different device.

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

If a user chooses not to encrypt an iTunes backup, the backup 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 is used for iTunes syncing and mobile device management (MDM). This keybag allows iTunes to back up and sync without requiring the user to enter a passcode, and it allows an MDM solution to remotely clear a user's passcode. It is stored on the computer that's used to sync with iTunes, or on the MDM solution that remotely manages the device.

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

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

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

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

The one-time Unlock Token for attended software updates expires after 20 minutes. This token is exported from the Secure Enclave and is written to Effaceable Storage. A policy timer increments the counter if the device hasn't rebooted within 20 minutes.

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

- Automatic updates are configured in iOS 12.

  or

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

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

After 8 hours a policy timer will invalidate the one-time Unlock Token.

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), so iCloud backups can be performed in the background. For all Data Protection classes except No Protection, the encrypted data is read from the device and sent to iCloud. The corresponding class keys are protected by iCloud keys. The Keychain class keys are wrapped with a UID-derived key in the same way as an unencrypted iTunes backup. An asymmetric keybag is also used for the backup in the Keychain recovery aspect of iCloud Keychain.

App Security

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

Because of this, iOS provides layers of protection to ensure that apps are signed and verified, and are sandboxed to protect user data. These elements provide a stable, secure platform for apps, enabling thousands of developers to deliver hundreds of thousands of apps on iOS without impacting system integrity. And users can access these apps on their iOS devices without undue fear of viruses, malware, or unauthorized attacks.

App code signing

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

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

iOS allows 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 will perform 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.

Businesses also have the ability to write in-house apps for use within their organization and distribute them to their employees. Businesses and organizations can apply to the Apple Developer Enterprise Program (ADEP) with a D-U-N-S number. Apple approves applicants after verifying their identity and eligibility. After an organization becomes a member of ADEP, it can register to obtain a Provisioning Profile that permits in-house apps to run on devices it authorizes. Users must have the Provisioning Profile installed to run the in-house apps. This ensures that only the organization's intended users are able to load the apps onto their iOS devices. Apps installed through 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.

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

Runtime process security

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

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

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

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

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

Extensions

iOS allows apps to provide functionality to other apps by providing *extensions*. Extensions are special-purpose signed executable binaries, packaged within an app. The system automatically detects extensions at install time and makes them available to other apps using a matching system.

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. The extension points are Today widgets, Share, Custom actions, Photo Editing, Document Provider, and Custom Keyboard.

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 will be extended to the extensions that are embedded within the app, but not to the extensions activated by the app.

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

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

App Groups

Apps and extensions owned by a given developer account can share content when configured to be part of an App Group. It is 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 guarantees that App Group IDs are unique across the app ecosystem.

Data Protection in apps

The iOS Software Development Kit (SDK) 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.

Accessories

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

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

Accessories can request access to different transport methods and functionality; for example, access to digital audio streams over the Lightning cable, or location information provided over Bluetooth. An authentication IC ensures that only approved accessories are granted full access to the device.

If an accessory doesn't support authentication, its access is limited to analog audio and a small subset of serial (UART) audio playback controls.

AirPlay also utilizes the authentication IC to verify that receivers have been approved by Apple. AirPlay audio and CarPlay video streams utilize the MFi-SAP (Secure Association Protocol), which encrypts communication between the accessory and device using AES-128 in CTR mode. Ephemeral keys are exchanged using ECDH key exchange (Curve25519) and signed using the authentication IC's 1024-bit RSA key as part of the Station-to-Station (STS) protocol.

HomeKit

HomeKit provides a home automation infrastructure that utilizes iCloud and iOS security to protect and synchronize private data without exposing it to Apple.

HomeKit identity

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

The keys are stored in Keychain and are included only in encrypted Keychain backups. The keys are synchronized between devices using iCloud Keychain where available. HomePod and Apple TV receive keys via tap-to-setup or the setup mode described below. Keys are shared from an iPhone to a paired Apple Watch via Apple identity service (IDS).

Communication with HomeKit accessories

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

To establish a relationship between an iOS 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 device by the user, and then encrypted using ChaCha20-Poly1305 AEAD with HKDF-SHA-512 derived keys. The accessory's MFi certification is also verified during setup. Accessories without an MFi chip can build in support for software authentication on iOS 11.3 or later.

When the iOS device and the HomeKit accessory communicate during use, each authenticates the other utilizing the keys exchanged in the above process. Each session is established using the Station-to-Station protocol and is encrypted with HKDF-SHA-512 derived keys based on per-session Curve25519 keys. This applies to both IP-based and Bluetooth Low Energy accessories.

For Bluetooth Low Energy devices that support broadcast notifications, the accessory is provisioned with a broadcast encryption key by a paired iOS device over a secure session. This key is used to encrypt the data about state changes on the accessory, which are notified via the Bluetooth Low Energy advertisements. The broadcast encryption key is an HKDF-SHA-512 derived key, and the data is encrypted using CHACHA20-POLY1305 Authenticated Encryption with Associated Data (AEAD) algorithm. The broadcast encryption key is periodically changed by the iOS device and synchronized to other devices using iCloud as described in the "Data synchronization between devices and users" section below.

Local data storage

HomeKit stores data about the homes, accessories, scenes, and users on a user's iOS device. This stored data is encrypted using keys derived from the user's HomeKit identity keys, plus a random nonce. Additionally, HomeKit data is stored using Data Protection class Protected Until First User Authentication. HomeKit data is only backed up in encrypted backups, so, for example, unencrypted iTunes backups don't contain HomeKit data.

Data synchronization between devices and users

HomeKit data can be synchronized between a user's iOS devices using iCloud and iCloud Keychain. The HomeKit data is encrypted during the synchronization using keys derived from the user's HomeKit identity and random nonce. This data is handled as an opaque blob during synchronization. The most recent blob is stored in iCloud to enable synchronization, but it isn't used for any other purposes. Because it is encrypted using keys that are available only on the user's iOS devices, its contents are inaccessible during transmission and iCloud storage.

HomeKit data is also synchronized between multiple users of the same home. This process uses authentication and encryption that is the same as that used between an iOS 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 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 devices that are part of the HomeKit home.

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

Home data and apps

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

HomeKit and Siri

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

HomeKit IP cameras

IP cameras in HomeKit send video and audio streams directly to the iOS device on the local network accessing the stream. The streams are encrypted using randomly generated keys on the iOS device and the IP camera, which are exchanged over the secure HomeKit session to the camera. When the iOS device isn't on the local network, the encrypted streams are relayed through the home hub to the iOS device. The home hub doesn't decrypt the streams and only functions as a relay between the iOS device and the IP camera. When an app displays the HomeKit IP camera video view to the user, HomeKit is rendering the video frames securely from a separate system process so the app is unable to access or store the video stream. In addition, apps aren't permitted to capture screenshots from this stream.

iCloud remote access for HomeKit accessories

HomeKit accessories can connect directly with iCloud to enable iOS devices to control the accessory when Bluetooth or Wi-Fi communication isn't available.

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

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

Accessories that support iCloud remote access are provisioned during the accessory's setup process. The provisioning process begins with the user signing in to iCloud. Next, the iOS 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 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 device that is conducting the provisioning also sends a bag to the accessory, which contains the URLs and other information needed to connect to the iCloud remote access server. This information isn't specific to any user or accessory.

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

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

Accessories connect to the iCloud remote access server using HTTP/2, secured using TLS v1.2 with AES-128-GCM and SHA-256. The accessory keeps its connection to the iCloud remote access server open so that it can receive incoming messages and send responses and outgoing notifications to iOS devices.

HomeKit TV Remote accessories

Third-party HomeKit TV Remote accessories provide HID events and Siri audio to an associated Apple TV added via the Home app. The HID events are sent over the secure session between the Apple TV and the Remote. A Siri-capable TV Remote sends audio data to the Apple TV when the user explicitly activates the microphone on the Remote using a dedicated Siri button. The audio frames are sent directly to the Apple TV using a dedicated local network connection between the Apple TV and the Remote. The local network connection is encrypted with a per-session HKDF-SHA-512 derived key-pair that is negotiated over the HomeKit session between the Apple TV and TV Remote. HomeKit decrypts the audio frames on the Apple TV and forwards them to the Siri app, where they are treated with the same privacy protections as all Siri audio input.

SiriKit

Siri utilizes the iOS Extension mechanism to communicate with third-party apps. Although Siri has access to iOS contacts and the device's current location, Siri checks the permission to access iOS-protected user data of the app containing the Extension to see if the app has access before providing that information to it. Siri passes only the relevant fragment of the original user query text to the extension. For example, if the app doesn't have access to iOS contacts, Siri won't resolve a relationship in a user request such as "Pay my mother 10 dollars using PaymentApp." In this case, the Extension's app would only see "mother" through the raw utterance fragment being passed to it. However, if the app does have iOS contacts access, it would receive the iOS contact information for the user's mother. If a contact were referred to in the body of a message—for example, "Tell my mother on Message app that my brother is awesome"—Siri wouldn't resolve "my brother" regardless of the app's TCCs. Content presented by the app may be sent to the server to allow Siri to understand vocabulary a user may use in the app.

In cases like "Get me a ride to my mom's home using <app name>"—where the user's request requires location information to be retrieved from the user's contacts, Siri provides that location information to the app's extension, for that request only, regardless of the app's location or contacts access.

At runtime, Siri allows the SiriKit-enabled app to provide a set of custom words specific to application instance. These custom words are tied to the random identifier discussed in the Siri section of this paper, and have the same lifetime.

HealthKit

HealthKit stores and aggregates data from health and fitness apps with permission of the user. 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.

Health data

HealthKit allows users to store and aggregate their health data from sources like apps, devices and healthcare institutions. This data is stored in Data Protection class Protected Unless Open. Access to the data is relinquished 10 minutes after device locks and data becomes accessible the next time user enters their passcode or uses Touch ID or Face ID to unlock the device.

HealthKit also aggregates management data, such as access permissions for apps, names of devices connected to HealthKit, and scheduling information used to launch apps when new data is available. This data is stored in Data Protection class Protected Until First User Authentication.

Temporary journal files store health records that are generated when the device is locked, such as when the user is exercising. These are stored in Data Protection class Protected Unless Open. When the device is unlocked, the temporary journal files are imported into the primary health databases, then deleted when the merge is completed.

Health data can be stored in iCloud. When configured for iCloud storage, Health data is synced between devices and secured by encryption that protects the data both in transit and at rest. Health data is only included in encrypted iTunes Backups. It isn't included in either unencrypted iTunes backups or iCloud Backup.

Clinical health records

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

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 IETF RFC 5652.

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 utilize 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 in the Sources tab of the Health app.

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

Medical ID

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

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

ReplayKit

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

Movie recording

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

- **Permissions dialog:** Before recording starts, ReplayKit presents a user consent alert requesting that the user acknowledge their intent to record the screen, the microphone, and the front-facing camera. This alert is presented once per app process, and will be re-presented if the app is left in the background for longer than 8 minutes.

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

Broadcasting

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

  The architecture ensures that hosting apps have no privileges to the broadcasted video and audio contents–only ReplayKit and the third-party broadcast extensions have access.
- **Broadcast picker:** To select which broadcast service to use, ReplayKit provides a view controller (similar to UIActivityViewController) that the developer can present in their app. The view controller is implemented using the UIRemoteViewController SPI and is an extension that lives within the ReplayKit framework. It is out-of-process from the hosting app.
- **System broadcast picker:** allows users initiate system broadcast directly from app using the same system defined UI that is accessible via Control Center. The UI is implemented using the UIRemoteViewController SPI and is an extension that lives within the ReplayKit framework. It is out-of-process from the hosting app.
- **Upload extension:** The upload extension that third-party broadcast services implement to handle video and audio content during broadcasting can choose to receive content in two ways:
  - Small encoded MP4 clips
  - Raw unencoded sample buffers
    - **MP4 clip handling:** During this mode of handling, the small encoded MP4 clips are generated by *replayd* and stored in a private location only accessible to ReplayKit's subsystems. After a movie clip is generated, *replayd* will pass the location of the movie clip to the third-party upload extension through the NSExtension request SPI (XPC based). *replayd* also generates a one-time sandbox token that's also passed to the upload extension, which grants the extension access to the particular movie clip during the extension request.
    - **Sample buffer handling:** 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 over through XPC to the third-party extension, and decoding securely back into an IOSurface object.

Secure Notes

The Notes app includes a Secure Notes feature that allows users to protect the contents of specific notes. Secure notes are encrypted using a user-provided passphrase that is required to view the notes on iOS, macOS, and the iCloud website.

When a user secures a note, a 16-byte key is derived from the user's passphrase using PBKDF2 and SHA256. The note's contents are encrypted using AES-GCM. New records are created in Core Data and CloudKit to store the encrypted note, tag, and initialization vector, and the original note records are deleted; the encrypted data isn't written in place. Attachments are also encrypted in the same way. Supported attachments 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.

When a user successfully enters the passphrase, whether to view or create a secure note, Notes opens a secure session. While open, the user isn't required to enter the passphrase—or use Touch ID or Face ID—to view or secure other notes. However, if some notes have a different passphrase, the secure session applies only to notes protected with the current 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.
- The device locks.

Users who forget their passphrase can still view secure notes or secure additional notes if they enabled Touch ID or Face ID on their devices. In addition, Notes will show a user-supplied hint after three failed attempts to enter the passphrase. The user must know the current passphrase in order to change it.

Users can reset the passphrase if they have forgotten the current one. 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 can be shared with others. Shared Notes aren't end-to-end encrypted. Apple uses 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/decrypt each other's data.

Apple Watch

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

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

Once the Bluetooth Low Energy session is established and encrypted using the highest security protocol available in Bluetooth Core Specification, Apple Watch and iPhone exchange keys using a process adapted from Apple identity service (IDS), as described under "iMessage" in the Internet Services section of this paper. After keys have been exchanged, the Bluetooth session key is discarded, and all communications between Apple Watch and iPhone are encrypted using IDS, with the encrypted Bluetooth, Wi-Fi, and Cellular links providing a secondary encryption layer. The Low Energy Bluetooth Address is rotated at 15-minute intervals to reduce the risk of traffic being compromised.

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

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

When Apple Watch isn't within Bluetooth range, Wi-Fi or cellular can be used instead. Apple Watch automatically joins Wi-Fi networks that have been already been joined on the paired iPhone and whose credentials have synced to the Apple Watch while both devices were in range. This Auto-Join behavior can then be configured on a per network basis in the Wi-Fi section of the Apple Watch's 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's Settings app.

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

Apple Watch can be manually locked by holding down the side button. Additionally, motion heuristics are used to attempt to automatically lock the device shortly after it's removed from the wrist. When Apple Watch is locked, Apple Pay can only be used by entering the watch’s passcode. Wrist detection is turned off using the Apple Watch app on iPhone. This setting can also be enforced using an MDM solution.

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

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

Apple Watch can be configured for a system software update the same night. For more information on how the Apple Watch passcode gets stored to be used during the update see the Keybags section of this document.

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

Network Security

In addition to the built-in safeguards Apple uses to protect data stored on iOS devices, there are many network security measures that organizations can take to keep information secure as it travels to and from an iOS device.

Mobile users must be able to access corporate networks from anywhere in the world, so it's important to ensure that they are authorized and their data is protected during transmission. iOS uses—and provides developer access to—standard networking protocols for authenticated, authorized, and encrypted communications. To accomplish these security objectives, iOS integrates proven technologies and the latest standards for both Wi-Fi and cellular data network connections.

On other platforms, firewall software is needed to protect open communication ports against intrusion. Because iOS achieves a reduced attack surface by limiting listening ports and removing unnecessary network utilities such as telnet, shells, or a web server, no additional firewall software is needed on iOS devices.

TLS

iOS supports Transport Layer Security (TLS v1.0, TLS v1.1, TLS v1.2) and DTLS. It supports both AES-128 and AES-256, and prefers cipher suites with perfect forward secrecy. Safari, Calendar, Mail, and other Internet apps automatically use this protocol to enable an encrypted communication channel between the device and network services. High-level APIs (such as CFNetwork) make it easy for developers to adopt TLS in their apps, while low-level APIs (Network.framework) provide fine-grained control. CFNetwork disallows SSLv3, and apps that use WebKit (such as Safari) are prohibited from making an SSLv3 connection.

As of iOS 11 and macOS High Sierra, 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 Sierra. By default, TLS clients or servers implemented with SecureTransport APIs don't have RC4 cipher suites enabled, and are unable to connect when RC4 is the only cipher suite available. To be more secure, services or apps that require RC4 should be upgraded to use modern, secure cipher suites.

App Transport Security

App Transport Security provides default connection requirements so that apps adhere to best practices for secure connections when using NSURLConnection, CFURL, or NSURLSession APIs. By default, App Transport Security limits cipher selection to include only suites that provide forward secrecy, specifically ECDHE_ECDSA_AES and ECDHE_RSA_AES in GCM or CBC mode. Apps are able to disable the forward secrecy requirement per-domain, in which case RSA_AES is added to the set of available ciphers.

Servers must support TLS v1.2 and forward secrecy, and certificates must be valid and signed using SHA-256 or better 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.

VPN

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

- IKEv2/IPSec with authentication by shared secret, RSA Certificates, **ECDSA** Certificates, EAP-MSCHAPv2, or EAP-TLS
- SSL-VPN using the appropriate client app from the App Store
- Cisco IPSec with user authentication by password and machine authentication by shared secret and certificates
- L2TP/IPSec with user authentication by MS-CHAPV2 password and machine authentication by shared secret

iOS supports 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. MDM can specify a connection for each managed app and specific domains in Safari. This helps ensure that secure data always goes to and from the corporate network—and that a user's personal data doesn't.
- **Always-on VPN,** which can be configured for devices managed through a mobile device management (MDM) solution and supervised using Apple Configurator 2, Apple School Manager, or Apple Business Manager. This eliminates the need for users to turn on VPN to enable protection when connecting to cellular and Wi-Fi networks. Always-on VPN gives an organization full control over device traffic by tunneling all IP traffic back to the organization. The default tunneling protocol, IKEv2, secures traffic transmission with data encryption. The organization can monitor and filter traffic to and from its devices, secure data within its network, and restrict device access to the Internet.

Wi-Fi

iOS supports industry-standard Wi-Fi protocols, including WPA2 Enterprise, to provide authenticated access to wireless corporate networks. WPA2 Enterprise uses AES-128 bit encryption, giving users the highest level of assurance that their data remains protected when sending and receiving communications over a Wi-Fi network connection. With support for 802.1X, iOS devices can be integrated into a broad range of RADIUS authentication environments. 802.1X wireless authentication methods supported on iPhone and iPad include EAP-TLS, EAP-TTLS, EAP-FAST, EAP-SIM, PEAPv0, PEAPv1, and LEAP.

Besides protection for data, iOS extends WPA2 level protection to unicast and multicast management frames through Protected Management Frame service referred in 802.11w. PMF support is available on iPhone 6 and iPad Air 2 or later.

iOS uses a randomized Media Access Control (MAC) address when conducting Wi-Fi scans while it isn't associated with a Wi-Fi network. These scans could be performed in order to find and connect a preferred 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.

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

Because a device's MAC address now changes when disconnected from a Wi-Fi network, it can't be used to persistently track a device by passive observers of Wi-Fi traffic, even when the device is connected to a cellular network. Apple has informed Wi-Fi manufacturers that iOS Wi-Fi scans use a randomized MAC address, and that neither Apple nor manufacturers can predict these randomized MAC addresses. Wi-Fi MAC address randomization support isn't available on iPhone 4s or earlier.

On iPhone 6S or later, the hidden property of a known Wi-Fi network is known and updated automatically. If the Service Set Identifier (SSID) of a Wi-Fi network is broadcasted, the iOS device won't send a probe with the SSID included in the request. This prevents the device from broadcasting the network name of non-hidden networks.

To protect the device from vulnerabilities in network processor firmware, network interfaces including Wi-Fi and baseband have limited access to application processor memory. When USB or SDIO is used to interface with the network processor, the network processor can't initiate Direct Memory Access (DMA) transactions to the application processor. When PCIe is used, each network processor is on its own isolated PCIe bus. An IOMMU on each PCIe bus limits the network processor's DMA access to pages of memory containing its network packets or control structures.

Bluetooth

Bluetooth support in iOS has been designed to provide useful functionality without unnecessary increased access to private data. iOS devices support Encryption Mode 3, Security Mode 4, and Service Level 1 connections. iOS supports the following Bluetooth profiles:

- Hands-Free Profile (HFP)
- Phone Book Access Profile (PBAP)
- Message Access Profile (MAP)
- Advanced Audio Distribution Profile (A2DP)
- Audio/Video Remote Control Profile (AVRCP)
- Personal Area Network Profile (PAN)
- Human Interface Device Profile (HID)

Support for these profiles varies by device.

For more information, go to:
https://support.apple.com/kb/ht3647

Single sign-on

iOS supports 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 (PKINIT) is also supported.

iOS SSO utilizes 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:

- 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 networking APIs can also be configured to use it. To configure SSO, iOS supports a configuration profile payload that allows 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.

Continuity

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

Handoff

With Handoff, when a user's Mac and iOS 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 4.2 pairing out-of-band using APNs. The individual messages are encrypted in a similar fashion to iMessage. After the devices are paired, each will generate a symmetric 256-bit AES key that gets stored in the device's Keychain. This key can encrypt and authenticate the Bluetooth Low Energy advertisements that communicate the device's current activity to other iCloud paired devices using AES-256 in GCM mode, with replay protection measures.

The first time a device receives an advertisement from a new key, it will establish a Bluetooth Low Energy connection to the originating device and perform an advertisement encryption key exchange. This connection is secured using standard Bluetooth Low Energy 4.2 encryption as well as encryption of the individual messages, which is similar to how iMessage is encrypted. In some situations, these messages will go via APNs instead of Bluetooth Low Energy. The activity payload is protected and transferred in the same way as an iMessage.

Handoff between native apps and websites

Handoff allows an iOS native app to resume web pages in domains legitimately controlled by the app developer. It also allows the native app user activity to be resumed in a web browser.

To prevent native apps from claiming to resume websites not controlled by the developer, the app must demonstrate legitimate control over the web domains it wants to resume. Control over a website domain is established via the mechanism for shared web credentials. For details, refer to "Access to Safari saved passwords" in the Encryption and Data Protection section of this paper. The system must validate an app's domain name control before the app is permitted to accept user activity Handoff.

The source of a web page Handoff can be any browser that has adopted the Handoff APIs. When the user views a web page, the system advertises the domain name of the web page in the encrypted Handoff advertisement bytes. Only the user's other devices can decrypt the advertisement bytes (as described earlier in this section).

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 web page. 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 will be 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 the basic feature of Handoff, some apps may elect to use APIs that support sending larger amounts of data over Apple-created peer-to-peer Wi-Fi technology (in a similar fashion to AirDrop). For example, the Mail app uses these APIs to support Handoff of a mail draft, which may include large attachments.

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

Universal Clipboard

Universal Clipboard leverages Handoff to securely transfer the content of a user's clipboard across devices so they can copy on one device and paste on another. Content is protected the same way as other Handoff data and 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 running on the user's other devices (as established by their iCloud sign-in).

Auto Unlock

Mac computers that support Auto Unlock use Bluetooth Low Energy and peer-to-peer Wi-Fi to securely allow the user's Apple Watch to unlock their Mac. Each capable Mac and Apple Watch associated with an iCloud account must use two-factor authorization (TFA).

When enabling an Apple Watch to unlock a Mac, a secure link using Auto Unlock Identities is established. The Mac creates a random one-time-use unlock secret and transmits it to the Apple Watch over the link. The secret is stored on Apple Watch and can only be accessed when Apple Watch is unlocked (see "Data Protection classes" in the Encryption and Data Protection section). Neither the master entropy nor the new secret is the user's password.

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

iPhone Cellular Call Relay

When a user's Mac, iPad, or iPod touch 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 will be notified via 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 will present the incoming call notification UI. Upon answering the call, the audio is seamlessly transmitted from the user's iPhone using a secure peer-to-peer connection between the two devices.

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

Outgoing calls are also relayed to iPhone via the Apple Push Notification service, and audio will be similarly transmitted over the secure peer-to-peer link between devices.

Users can disable phone call relay on a device by turning off iPhone Cellular Calls in FaceTime settings.

iPhone Text Message Forwarding

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

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

Instant Hotspot

iOS devices that support Instant Hotspot use Bluetooth Low Energy to discover and communicate to devices that have signed in to the same iCloud account. Compatible Mac computers running OS X Yosemite or later use the same technology to discover and communicate with Instant Hotspot iOS devices.

When a user enters Wi-Fi Settings on the iOS device, the device emits a Bluetooth Low Energy 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 availability.

When a user chooses a device available 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 standard Bluetooth Low Energy encryption, and the request is encrypted in a fashion similar to iMessage encryption. The device then responds across the same Bluetooth Low Energy link using the same per-message encryption with Personal Hotspot connection information.

AirDrop security

iOS 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 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 Wi-Fi Access Point.

When a user enables AirDrop, a 2048-bit RSA identity is stored on the device. Additionally, an AirDrop 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 device emits an AirDrop signal over Bluetooth Low Energy. Other devices that are awake, in close proximity, and have AirDrop turned on detect the signal and respond with a shortened version of their owner's identity hash.

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. In Contacts Only mode, the received identity hashes are compared with hashes of people in the initiator's Contacts app. If a match is found, the sending device creates a peer-to-peer Wi-Fi network and advertises an AirDrop connection using Bonjour. Using this connection, the receiving devices send their full identity hashes to the initiator. If the full hash still matches Contacts, the recipient's first name and photo (if present in Contacts) are displayed in the AirDrop share sheet.

When using AirDrop, the sending user selects who they want to share with. 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. Then the receiving user is asked to accept the incoming transfer from the identified person or device. If multiple recipients have been selected, this process is repeated for each destination.

In Everyone mode, the same process is used but if a match in Contacts isn't found, the receiving devices are shown in the AirDrop send sheet with a silhouette with the device's name, as defined in Settings > General > About > Name.

Organizations can restrict the use of AirDrop for devices or apps being managed by using an MDM solution.

Wi-Fi password sharing

iOS 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 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 Bluetooth Low Energy 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 identify is proven, the grantor sends the requestor the 64-character PSK, which can also be used to join the network.

Organizations can restrict the use of Wi-Fi password sharing for devices or apps being managed by using an MDM solution.

Apple Pay

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

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

Apple Pay components

Secure Element: The Secure Element is an industry-standard, certified chip running the Java Card platform, which is compliant with financial industry requirements for electronic payments.

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.

Wallet: 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 Wallet. Users can also add cards to Apple Pay in:

- Setup Assistant and Settings for iOS
- The Watch app for Apple Watch
- The Wallet & Apple Pay system preference pane for Mac.

In addition, 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, and Apple Watch, 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 to the Secure Element or Secure Enclave where available, directly 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 Wallet and 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 re-encrypting payment credentials for payments within apps.

How Apple Pay uses the 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.

How Apple Pay uses the NFC controller

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

After a credit, debit, or prepaid card (including store cards) payment is authorized by the cardholder using Touch ID, Face ID, or 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 card provisioning

When a user adds a credit, debit, or prepaid card (including store cards) to 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 will determine whether to approve adding the card to Wallet.

Apple Pay uses three server-side calls to send and receive communication with the card issuer or network as part of the card provisioning process: *Required Fields*, *Check Card,* and *Link and Provision*. The card issuer or network uses these calls to verify, approve, and add cards to Wallet. These client-server sessions are encrypted using TLS v1.2.

Full card numbers aren't stored on the device or on Apple 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 usual 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 isolated from iOS and watchOS, is never stored on Apple servers, and is never backed up to iCloud.

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 will 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 seven days. This feature requires two-factor authentication to be enabled on the respective iCloud account.

Adding a credit or debit card manually to Apple Pay

To add a card manually, the name, card number, expiration date, and CVV are used to facilitate the provisioning process. From within Settings, the Wallet app, or the Apple Watch app, users can enter that information by typing, or using the camera on the device. 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 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 your iTunes and App Store account activity (for example, whether you have a long history of transactions within iTunes), information about your device (for example, phone number, name, and model of your device plus any companion iOS device necessary to set up Apple Pay), as well as your approximate location at the time you add your card (if you have Location Services enabled). Using this information, the card issuer will determine 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 to Apple Pay

For a credit or debit card on file with iTunes, the user may be required to re-enter 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 will download and display terms and conditions, then send along the term's ID and the card security code to the Link and Provision process. Additional verification may occur for iTunes account cards on file.

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

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

Additional verification

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

Payment authorization

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

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

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

Credit, debit, and prepaid cards added to the Secure Element can only be used if the Secure Element is presented with authorization using the same pairing key and AR value from when the card was added. This allows iOS to instruct the Secure Enclave to render cards unusable by marking its copy of the AR as invalid under the following scenarios:

- When the passcode is disabled.
- The user signs out of iCloud.
- The user selects Erase All Content and Settings.
- The device is restored from recovery mode.

With Apple Watch, cards are marked as invalid when:

- The watch’s passcode is disabled.
- The watch is unpaired from iPhone.

Using the pairing key and its copy of the current AR value, the Secure Element verifies the authorization received from the Secure Enclave before enabling the payment applet for a contactless payment. This process also applies when retrieving encrypted payment data from a payment applet for transactions within apps.

Transaction-specific dynamic security code

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 that is incremented for each new transaction and a key that’s 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 the following:

- A Terminal Unpredictable Number in the case of an NFC transaction
- An Apple Pay Server nonce in the case of transactions within apps

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

Paying with credit and debit cards in stores

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

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

Next, the user must authenticate using Touch ID, Face ID, or their passcode before payment information is transmitted. 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 credit and debit cards within apps

Apple Pay can also be used to make payments within iOS apps and Apple Watch apps. When users pay within apps using Apple Pay, Apple receives encrypted transaction information and re-encrypts it with a developer-specific key before it's sent to the developer or merchant. 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 re-encrypt it with a merchant-specific key before relaying the transaction to the merchant.

When an app requests a payment, it calls an API to determine if the device supports Apple Pay and if the user has credit or debit cards that can make payments on a payment network accepted by the merchant. The app requests any pieces of information it needs to process and fulfill the transaction, such as the billing and shipping address, and contact information. The app then asks iOS 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 will be transferred to the merchant.

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 will be 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 re-encrypt the payment credential with the merchant key associated with the Merchant ID. It's 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 to send to the Secure Element to be signed, such as an order number or customer identity, ensuring 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 credit and debit cards on the web

Apple Pay can be used to make payments on websites with iOS devices, Apple Watch, and Mac. Apple Pay transactions can also start on a Mac and be completed on an Apple Pay enabled iPhone or Apple Watch using the same iCloud account.

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

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

In the case of Mac to iPhone or Apple Watch Handoff, Apple Pay 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, passcode, or double-clicks 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 advertisements.

Contactless passes

Wallet supports the value added service (VAS) protocol for transmitting data from supported passes to compatible NFC terminals. 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 merchant's identifier, the user is asked to authorize its use using Touch ID, Face ID, or passcode. The pass information, a timestamp, and a single-use random ECDH P-256 key is used with the merchant's public key to derive an encryption key for the pass data, which is sent to the terminal.

Users may also manually select a pass and authorize it using Touch ID, Face ID, or passcode, before presenting it to the merchant's NFC terminal.

Apple Pay Cash

As of iOS 11.2 and watchOS 4.2, 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 Pay Cash account that can be accessed in Wallet or within Settings > Wallet & Apple Pay across any of the eligible devices the user has signed in with their Apple ID.

To use person-to-person payments and Apple Pay Cash, a user must be signed in to their iCloud account on an Apple Pay Cash compatible device, and have two-factor authentication set up on the iCloud account.

When you set up Apple Pay Cash, the same information as when you add 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 your privacy by storing and processing information separately from the rest of Apple, and in a way that the rest of Apple doesn't know. This information is used only for troubleshooting, fraud prevention, and regulatory purposes.

Money requests and transfers between users are initiated from within the Messages app or by asking Siri. When a user attempts to send money, iMessage will display the Apple Pay sheet. The Apple Pay Cash balance is always used first. If necessary, additional funds are drawn from a second credit or debit card the user has added to Wallet.

The Apple Pay Cash card in Wallet can be used with Apple Pay to make payments in stores, in apps, and on the web. Money in the Apple Pay 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 Pay Cash account from a debit or prepaid card in Wallet.

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

When you send money with Apple Pay, add money to an Apple Pay Cash account, or transfer 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 you are notified of a completed transaction.

If the transaction involves a credit or debit card for adding money to Apple Pay Cash, sending money to another user, or providing supplemental money if the Apple Pay Cash balance is insufficient, then an encrypted payment credential is also produced and sent to Apple Pay Servers, similar to what is used for Apple Pay within apps and websites.

After the balance of the Apple Pay Cash account exceeds a certain amount, or if unusual activity is detected, the user will be prompted to verify their identity. Information provided to verify the user's identity, such as social security number or answers to questions (for example, confirm street name you have previously lived on), is securely transmitted to Apple's partner and encrypted using their key. Apple can't decrypt this data.

Transit cards

In China and Japan, users can add supported transit cards to Wallet on supported models of iPhone and Apple Watch. This can be done either by transferring the value and commuter pass from a physical card into its digital Wallet representation or by provisioning a new transit card into Wallet from the transit card issuer's app. After transit cards are added to Wallet, users can ride transit simply by holding iPhone or Apple Watch near the transit reader. In Japan, the Suica card 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 Wallet, Apple or the transit card issuer may be able to link the user's personal information and the associated account information between cards. For example, MySuica cards can be linked to anonymous Suica cards. Transit cards and transactions are protected by a set of hierarchical cryptographic keys.

During the process of transferring the balance from physical card to Wallet, users are required to enter identifying digits of the card's serial number. Users may also need to provide personal information for proof of card possession. For example, if the card is a MySuica card or a Suica card that contains a commuter pass, users must also enter their date of birth. When transferring passes from iPhone to Apple Watch, both devices must be online during transfer.

The balance can be recharged with funds from credit and prepaid cards through Wallet or from the transit card issuer's app. The security of reloading the balance when using Apple Pay is described in the "Paying with Credit and Debit cards within apps" section of this paper.

The process of provisioning the transit card from within the transit card issuer's app is described in the "Adding credit or debit cards from a card issuer's app" section of this paper.

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

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

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

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

- Erasing the device remotely with Find My iPhone
- Enabling Lost Mode with Find My iPhone
- 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 Wallet
- Removing the card in the issuer's app

Apple Pay Servers notify the transit operator to suspend or disable those cards. For Suica cards, if users' devices are offline when they try to erase them, their Suica cards might still be available for use at some terminals until 12:01 a.m. JST the following day. If the users' device is offline, transit cards in China continue to be available for use.

If users remove their transit cards, the balance is recoverable by adding them back to a device signed in with the same Apple ID.

Student ID cards

In iOS 12, students, faculty, and staff at participating campuses can add their ID card to Wallet to access locations and pay wherever their card is accepted.

A user adds their ID card to Wallet through an app provided by the ID card issuer or participating school. The technical process by which this occurs is the same as the one described in the section "Adding credit or debit cards from a card issuer's app" earlier in this guide. In addition, issuing apps must support two-factor authentication on the accounts that guard access to their 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 Wallet, Express Mode is turned on by default. Student ID cards in Express Mode interact with accepting terminals without Touch ID, Face ID, or passcode authentication. The user can tap the More button on the front of the ID card in Wallet and turn off Express Mode to disable this feature. Touch ID, Face ID, or passcode is required to re-enable Express Mode.

Student ID cards can be disabled or removed by:

- Erasing the device remotely with Find My iPhone
- Enabling Lost Mode with Find My iPhone
- 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 Wallet
- Removing the card in the issuer's app

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 iPhone. Users also have the ability to remove and erase their cards from Apple Pay using Find My iPhone, iCloud.com, or directly on their devices using Wallet. 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 will be suspended or removed from Apple Pay by the card issuer or respective payment network, even if the device is offline and not connected to a cellular or Wi-Fi network. Users can also call their card issuer to suspend or remove cards from Apple Pay.

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

Internet Services

Apple has built a robust set of services to help users get even more utility and productivity out of their devices, including iMessage, FaceTime, Siri Suggestions, iCloud, iCloud Backup, and iCloud Keychain.

These Internet services have been built with the same security goals that iOS promotes throughout the platform. These goals include secure handling of data, whether at rest on the device or in transit over wireless networks; protection of users' personal information; and threat protection against malicious or unauthorized access to information and services. Each service uses its own powerful security architecture without compromising the overall ease of use of iOS.

Creating strong Apple ID passwords

Apple IDs are used to connect to a number of services including iCloud, FaceTime, and iMessage. To help users create strong passwords, all new accounts must contain the following password attributes:

- At least eight characters
- At least one letter
- At least one uppercase letter
- At least one number
- No more than three consecutive identical characters
- Not the same as the account name

Apple ID

An Apple ID is the account that is used to sign in to Apple services such as iCloud, iMessage, FaceTime, the iTunes Store, Apple Books, the App Store, and more. It is important for users to keep their Apple IDs secure to prevent unauthorized access to their accounts. To help with this, Apple requires strong passwords that must be at least eight characters in length, contain both letters and numbers, must not contain more than three consecutive identical characters, and 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 requires users to set up three security questions that can be used to help verify the owner's identity when making changes to their account information or resetting a forgotten password.

Apple also sends email and push notifications to users 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 customer security.

Two-factor authentication

To help users further secure their accounts, Apple offers *two-factor authentication*—an extra layer of security for Apple IDs. It is designed to ensure that only the account's owner can access the account, even if someone else knows the password.

With two-factor authentication, a user's account can be accessed only on trusted devices, such as the user's iPhone, iPad, or Mac. 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 automatically displayed on the user's trusted devices or sent to a trusted phone number. By entering the code, the user verifies that they trust the new device and that it's safe to sign in.

Because a password alone is no longer enough to access a user's account, two-factor authentication improves the security of the user's Apple ID and all the personal information they store with Apple. It is integrated directly into iOS, macOS, tvOS, watchOS, and the authentication systems used by Apple's websites.

For more information on two-factor authentication, go to:
https://support.apple.com/HT204915

Two-step verification

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

For more information on two-step verification for Apple ID, go to:
https://support.apple.com/kb/ht5570

Managed Apple IDs

Managed Apple IDs function in a way similar to an Apple ID, but are owned and controlled by an educational institution. The institution can reset passwords, limit purchasing and communications such as FaceTime and Messages, and set up role-based permissions for staff members, teachers, and students.

Some Apple services are disabled for Managed Apple IDs, such as Apple Pay, iCloud Keychain, HomeKit, and Find My iPhone.

For more information about Managed Apple IDs, go to:
https://help.apple.com/schoolmanager/#/tes78b477c81

Auditing Managed Apple IDs

Managed Apple IDs also support auditing, which allows institutions to comply with legal and privacy regulations. Administrator, manager, or teacher accounts can be granted auditing privileges for specific Managed Apple IDs. Auditors can monitor only accounts that are below them in the school's hierarchy. That is, teachers can monitor students; managers can audit teachers and students; and administrators can audit managers, teachers, and students.

When auditing credentials are requested using Apple School Manager, a special account is issued that has access only to the Managed Apple ID for which auditing was requested. Auditing permission expires after seven days. During that period, the auditor can read and modify the user's content stored in iCloud or CloudKit-enabled apps. Every request for auditing access is logged in Apple School Manager. The logs show who the auditor was, the Managed Apple ID the auditor requested access to, the time of the request, and if the auditing was performed.

For more information about auditing Managed Apple IDs, go to:
https://help.apple.com/schoolmanager/#/tesd8fcbdd99

Managed Apple IDs and personal devices

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

iMessage

Apple iMessage is a messaging service for iOS devices, Apple Watch, and Mac computers. iMessage supports text and attachments such as photos, contacts, and locations. 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 two pairs of keys for use with the service: an RSA 1280-bit key for encryption and an ECDSA 256-bit key on the NIST P-256 curve for signing. The private keys for both key pairs are saved in the device's Keychain and 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 will be presented with a confirmation dialog if the SMS is not zero rated). Phone number verification may be required for several system services in addition to iMessage, such as FaceTime and iCloud. All of the user's registered devices display an alert message when a new device, phone number, or email address is added.

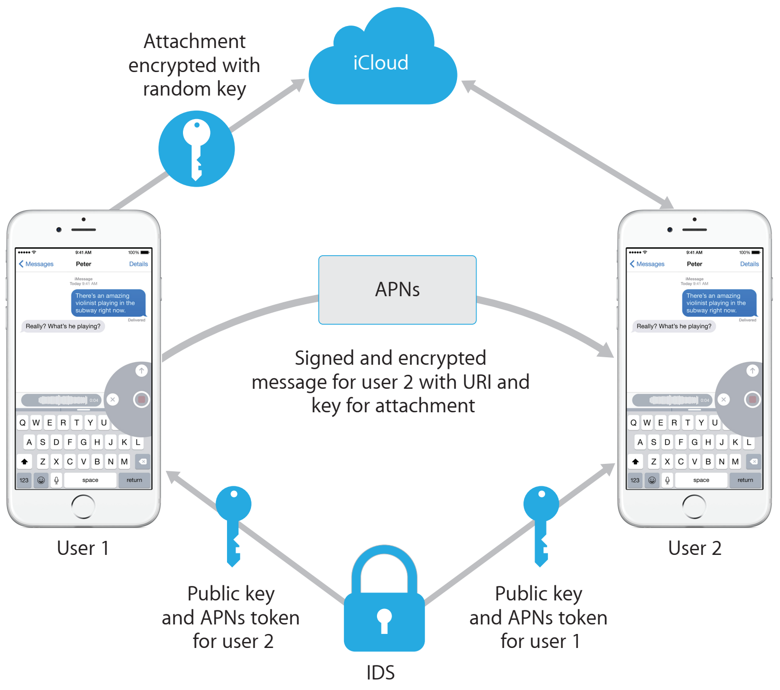
In iOS 12, messages sent from different addresses but that are linked to the same Apple ID appear as a single conversation on devices that receive them. This is facilitated by an account identifier retrieved from IDS alongside the public keys and APNs addresses for an email address or phone number.

How iMessage sends and receives messages

Users start a new iMessage conversation by entering an address or name. If they enter a phone number or email address, the device contacts the 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 utilizes the user's Contacts app to gather the phone numbers and email addresses associated with that name, then gets the public keys and APNs addresses from IDS.

The user's outgoing message is individually encrypted for each of the receiver's devices. The public RSA encryption keys of the receiving devices are retrieved from IDS. For each receiving device, the sending device generates a random 88-bit value and uses it as an HMAC-SHA256 key to construct a 40-bit value derived from the sender and receiver public key and the plaintext. The concatenation of the 88-bit and 40-bit values makes a 128-bit key, which encrypts the message with it using AES in CTR mode. The 40-bit value is used by the receiver side to verify the integrity of the decrypted plaintext. This per-message AES key is encrypted using RSA-OAEP to the public key of the receiving device. The combination of the encrypted message text and the encrypted message key is then hashed with SHA-1, and the hash is signed with ECDSA using the sending device's private signing key. The resulting messages, one for each receiving device, consist of the encrypted message text, the encrypted message key, and the sender's digital signature. They are then dispatched to the APNs for delivery. Metadata, such as the timestamp and APNs routing information, isn't encrypted. Communication with APNs is encrypted using a forward-secret TLS channel.

APNs can only relay messages up to 4KB or 16KB in size, depending on iOS 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 **URI (Uniform Resource Identifier)**, and a SHA-1 hash of its encrypted form are then sent to the recipient as the contents of an iMessage, with their confidentiality and integrity protected through normal iMessage encryption, as shown in the following diagram.


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

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 is delivered. Unlike other APNs notifications, however, iMessage messages are queued for delivery to offline devices. Messages are currently stored for up to 30 days.

Business Chat

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

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

FaceTime

FaceTime is Apple's video and audio calling service. Similar to iMessage, FaceTime calls also use the Apple Push Notification service 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 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 via the Secure Real-time Transport Protocol (SRTP). SRTP packets are encrypted using AES-256 in Counter Mode and HMAC-SHA1. Subsequent to the initial connection and security setup, FaceTime uses STUN and Internet Connectivity Establishment (ICE) to establish a peer-to-peer connection between devices, if possible.

iCloud

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. iCloud features, including My Photo Stream, iCloud Drive, and iCloud Backup, can be disabled by IT administrators via MDM configuration profiles. The service is agnostic about what is being stored and handles all file content the same way, as a collection of bytes.

Each file is broken into chunks and encrypted by iCloud using AES-128 and a key derived from each chunk's contents that utilizes SHA-256. The keys and the file's metadata are stored by Apple in the user's iCloud account. The encrypted chunks of the file are stored, without any user-identifying information or the keys, using both Apple and third-party storage services.

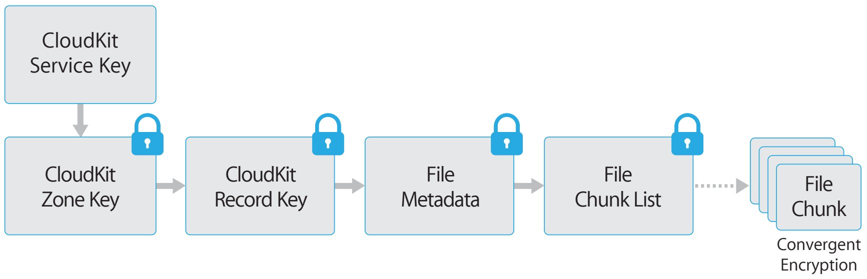
iCloud Drive

iCloud Drive adds account-based keys to protect documents stored in iCloud. As with existing iCloud services, it 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 must also possess the iCloud Drive Service Key to expose protected parts of iCloud Drive storage.

CloudKit

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

As with iCloud Drive, CloudKit uses account-based keys to protect the information stored in the user's private database and, similar to other iCloud services, files are chunked, encrypted, and stored using third-party services. CloudKit utilizes 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

Apple Pay Cash, Health data, User keywords, Siri Intelligence, and Hey Siri use CloudKit 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—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 (see "Escrow security" section later in paper), the data in CloudKit is reset; and if data is available from the trusted local device, it is re-uploaded to CloudKit.

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.

iCloud 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 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, the system is designed to keep data secure while allowing incremental, unattended backup and restoration to occur.

Here's what iCloud backs up:

- 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 iOS 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, Apple Books, or the App Store. 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 iOS devices. Note that if a user turns on iCloud Photo Library on their iOS device (iOS 8.1 or later) or Mac (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
- Call history and ringtones
- Home screen and app organization
- HomeKit configuration
- HealthKit 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)

  Note: When Messages in the Cloud is enabled, iMessage, Business Chat, text (SMS), and MMS messages are removed from the user's existing iCloud Backup, and are instead stored in an end-to-end encrypted CloudKit container for Messages. The user's iCloud Backup retains a key to that container. If the user subsequently disables iCloud Backup, that container's key is rolled, the new key is only stored 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.

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. Files are backed up to iCloud in their original, encrypted state. Files in Data Protection class No Protection are encrypted during transport.

Recovery Options

Situation	User recovery options for CloudKit end-to-end encryption
Access to trusted device	Data recovery possible via a trusted device or iCloud Keychain recovery.
No trusted devices	Data recovery only possible via iCloud Keychain recovery.

Situation	User recovery options for Messages in iCloud
iCloud Backup enabled and access to trusted device	Data recovery possible via iCloud Backup, access to a trusted device or iCloud Keychain recovery.
iCloud Backup enabled and no access to trusted device	Data recovery possible via iCloud Backup and iCloud Keychain recovery.
iCloud Backup disabled and access to trusted device	Data recovery possible via a trusted device or iCloud Keychain recovery.
Backup disabled and no trusted devices	Data recovery only possible via iCloud Keychain recovery.

The iCloud Backup keybag contains asymmetric (Curve25519) keys for each Data Protection class, which are used to encrypt the per-file keys. For more information about the contents of the backup keybag and the iCloud Backup keybag, see “Keychain Data Protection” in the Encryption and Data Protection section of this paper.

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 utilized 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 re-encrypting them as per their Data Protection class.

iCloud Keychain

Safari integration with iCloud Keychain
Safari can automatically generate cryptographically strong random strings for website passwords, which are stored in Keychain and synced to other devices. Keychain items are transferred from device to device, traveling through Apple servers, but are encrypted in such a way that Apple and other devices can’t read their contents.

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

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

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

Keychain syncing

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

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

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

When the first device sees that an application ticket has arrived, it displays a notice 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. This establishes that the person who generated the request to join the circle entered the user's iCloud password at the time the request was made.

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, 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 is 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 via iCloud key value storage or store them in CloudKit as appropriate. If both circle members have the same item, the one with the most recent modification date will be synced. Items are skipped if the other member has the item and the modification dates are identical. Each item that's synced is encrypted so it can be decrypted only by a device within the user's circle of trust. It can't be decrypted by any other devices or Apple.

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

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

Keychain recovery

Keychain recovery provides a way for users to optionally escrow their Keychain with Apple, without allowing Apple to read the passwords and other data it contains. Even if the user has only a single device, Keychain recovery provides a safety net against data loss. This is particularly important when Safari is used to generate random, strong passwords for web accounts, as 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 will provide a copy of the Keychain only if a strict set of conditions are met.

When iCloud Keychain is turned on, if two-factor authentication is enabled for the user's account, the device passcode will be 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 let their devices create a cryptographically random code that they can record and keep on their own.

Next, the iOS device exports a copy of the user's Keychain, encrypts it wrapped with keys in an asymmetric keybag, and places it in the user's iCloud key value storage area. The keybag is wrapped with the user's iCloud Security Code and the public key of the hardware security module (HSM) cluster that will store the escrow record. This becomes the user's iCloud Escrow Record.

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

In addition to establishing a security code, users must register a phone number. This provides a secondary level of authentication during Keychain recovery. The user will receive an SMS that must be replied to in order for the recovery to proceed.

Escrow security

iCloud provides a secure infrastructure for Keychain escrow that ensures only authorized users and devices can perform a recovery. Topographically positioned behind iCloud are HSM clusters that guard the escrow records. Each has a key that is used to encrypt the escrow records under their watch, as described previously in this paper.

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

Next, the device uses the iCloud Security Code to unwrap the random key used to encrypt the user's Keychain. With that key, the Keychain—retrieved from iCloud key value storage—is decrypted and restored onto the device. Only 10 attempts to authenticate and retrieve an escrow record are allowed. 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 will cause the HSM cluster to delete the private key. Should this occur, the owner of each Keychain protected by the cluster will receive a message informing them that their escrow record has been lost. They can then choose to re-enroll.

Siri

By simply talking naturally, users can enlist Siri to send messages, schedule meetings, place phone calls, and more. Siri uses speech recognition, text-to-speech, and a client-server model to respond to a broad range of requests. The tasks that Siri supports have been designed to ensure that only the absolute minimal amount of personal information is utilized and that it is fully protected.

When Siri is turned on, the device creates random identifiers for use with the voice recognition and Siri servers. These identifiers are used only within Siri and are utilized to improve the service. If Siri is subsequently turned off, the device will generate a new random identifier to be used if Siri is turned back on.

To facilitate Siri features, some of the user's information from the device is sent to the server. This includes information about the music library (song titles, artists, and playlists), the names of Reminders lists, and names and relationships that are defined in Contacts. All communication with the server is over HTTPS.

When a Siri session is initiated, the user's first and last name (from Contacts), along with a rough geographic location, are sent to the server. This allows Siri to respond with the name or answer questions that need only an approximate location, such as those about the weather.

If a more precise location is necessary—such as determining the location of nearby movie theaters—the server asks the device to provide a more exact location. This is an example of how, by default, information is sent to the server only when it's strictly necessary to process the user's request. In any event, session information is discarded after 10 minutes of inactivity.

When Siri is used from Apple Watch, the watch creates its own random unique identifier, as described previously. However, instead of sending the user's information again, its requests also send the Siri identifier of the paired iPhone to provide a reference to that information.

The recording of the user's spoken words is sent to Apple's voice recognition server. If the task involves dictation only, the recognized text is sent back to the device. Otherwise, Siri analyzes the text and, if necessary, combines it with information from the profile associated with the device. For example, if the request is "send a message to my mom," the relationships and names that were uploaded from Contacts are utilized. The command for the identified action is then sent back to the device to be carried out.

Many Siri functions are accomplished by the device under the direction of the server. For example, if the user asks Siri to read an incoming message, the server simply tells the device to speak the contents of its unread messages. The contents and sender of the message aren't sent to the server.

User voice recordings are saved for a six-month period so that the recognition system can utilize them to better understand the user's voice. After six months, another copy is saved, without its identifier, for use by Apple in improving and developing Siri for up to two years. A small subset of recordings, transcripts, and associated data without identifiers may continue to be used by Apple for ongoing improvement and quality assurance of Siri beyond two years. Additionally, some recordings that reference music, sports teams and players, and businesses or points of interest are similarly saved for purposes of improving Siri.

Siri can also be invoked hands-free via voice activation. The voice trigger detection is performed locally on the device. In this mode, Siri is activated only when the incoming audio pattern sufficiently matches the acoustics of the specified trigger phrase. When the trigger is detected, the corresponding audio including the subsequent Siri command is sent to Apple's voice recognition server for further processing, which follows the same rules as other user voice recordings made through Siri.

Users can also invoke Siri on Apple Watch by raising their watch close to their mouth and speaking a Siri request. Siri is activated in this way when both:

- An on-device machine learning model detects the acoustics of human speech near the device
- A second on-device machine learning model identifies a motion profile and device pose matching the Raise to Speak gesture

When this combination of motion and audio is detected, the corresponding audio is sent to Apple's voice recognition server for further processing, which follows the same rules as other user voice recordings made through Siri.

Siri Suggestions

Siri suggestions for apps and shortcuts are generated using on-device machine learning. No data goes to Apple except information which can't be used to identify the user about what signals were useful predictors of shortcuts or app launches.

Shortcuts in Siri

Shortcuts added to Siri are synced across all Apple devices using iCloud, and encrypted using CloudKit end-to-end encryption. The phrases associated with shortcuts are synced to the Siri server for speech recognition, and associated with the random Siri identifier described in the Siri section. Apple doesn't receive the contents of the shortcuts, which are stored locally in a data vault.

Shortcuts app

Custom shortcuts in the Shortcuts app are optionally synced across Apple devices using iCloud. Shortcuts can also be shared with other users through iCloud.

Custom shortcuts are versatile—they're similar to scripts or programs. A quarantine system is used to isolate shortcuts that were downloaded from the internet. The user is warned the first time they try to use the shortcut and given an opportunity to inspect the shortcut, including information about where it originated.

Custom shortcuts can also run user-specified JavaScript on websites in Safari when invoked from the share sheet. In order to protect against malicious JavaScript that, for example, trick the user into running a script on a social media website that harvests their data, updated malware definitions are downloaded to identify malicious scripts at run-time. The first time that 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.

Safari Suggestions, Siri Suggestions in Search, Lookup, #images, News app, and News widget in non-News countries

Safari Suggestions, Siri Suggestions in Search, Lookup, #images, News app, and News widget in non-News countries show users suggestions that go beyond their devices, from sources like Wikipedia, the iTunes Store, local News, Maps results, and the App Store—and even offer suggestions before a user begins typing.

When a user starts typing in the Safari address bar, opens or uses Siri Suggestions in Search, uses Lookup, opens #images, uses Search in the News app, or uses the News widget in non-News countries, the following context is sent encrypted using HTTPS to Apple to provide the user with relevant results:

- An identifier that rotates every 15 minutes to preserve privacy.
- User's search query.
- The most likely query completion based on context and locally cached past searches.
- The approximate location of their device, if they have Location Services for Location-Based Suggestions turned on. The level of location "blurring" is based on estimated population density at the device's location; for instance, more blurring in a rural location where users may be geographically more separated versus less blurring in a city center where users will typically be closer together. Users can disable the sending of all location information to Apple in Settings, by turning off Location Services for Location-Based Suggestions. If Location Services is turned off, then Apple may use the device's IP address to infer an approximate location.
- The type of device and whether the search is made in Siri Suggestions in Search, Safari, Lookup, News app, or Messages.
- The type of connection.
- Information on the three apps most recently used on the device (for additional search context). Only apps that are in an Apple-maintained allow list of popular apps and have been accessed within the last 3 hours are included.
- A list of popular applications on the device.
- Regional language, locale, and input preferences.
- If the user's device can access music or video subscription services, then information such as names of the subscription services and types of subscriptions may be sent to Apple. The user's account name, number, and password aren't sent to Apple.
- Summarized, aggregated representation of topics of interests.

When a user selects a result or exits the app with no result selected, some information is sent to Apple to help improve the quality of future results. This information is tied only to the same 15-minute session identifier and not tied to a particular user. The feedback includes some of the previously described context information as well as interaction information such as:

- Timings between interactions and search network requests.
- Ranking and display order of suggestions.
- The ID of the result and action selected if result is non-local, or the category of the result selected if it is local.
- A flag indicating whether the user selected the result.

Apple retains Suggestions logs with queries, context, and feedback for 18 months. A subset of logs are retained for up to five years; for example queries, locale, domain, approximate location, and aggregate metrics.

In some cases, Suggestions may forward queries for common words and phrases to a qualified partner, in order to receive and display the partner's search results. Apple proxies the queries so that partners don't receive user IP addresses or search feedback. Communication with the partner is encrypted via HTTPS. For queries that occur frequently, Apple provides city-level location, device type, and client language as search context to the partner to improve search performance.

To understand and improve Suggestions performance geographically and across different types of networks, the following information is logged without a session identifier:

- Partial IP address (without the last octet for IPv4 addresses; without the last 80 bits for IPv6 addresses)
- Approximate location
- Approximate time of the query
- Latency/transfer rate
- Response size
- Connection type
- Locale
- Device type and requesting app

Safari Intelligent Tracking Prevention

Intelligent Tracking Prevention, or ITP, is part of Safari's privacy-friendly default cookies and website data policy. It helps prevent cross-site tracking by limiting access to cookies and other website data.

ITP collects statistics on resource loads (images, scripts etc.) as well as user interactions such as taps and text entries. A machine learning model is used for on-device classification of which domain names have the ability to track the user cross-site, based on the collected statistics.

When a domain is classified as having tracking abilities, ITP immediately partitions its cookies if the user has previously interacted with that domain as a first party; for classified domains that a user hasn't interacted with, ITP immediately begins to block their cookies. For example:

- video.example offers an ad-free subscription service and has many of its videos embedded on other websites
- A user signs in to video.example and then other websites containing video.example embedded content

- ITP classifies video.example as having tracking abilities and therefore partitions its cookies
- When a user visits newspaper.example and it contains embedded content from video.example, the cookies provided to video.example are partitioned cookies specific to video.example on newspaper.example

Embedded third-party content may ask a user for access to their first-party cookies with the Storage Access API. When a user taps or clicks on embedded third-party content that uses the Storage Access API, Safari presents a prompt asking if the user wants to allow the third-party to have access to its cookies and website data, which allows the third-party to track them on the first-party domain. If a user selects Allow, the embedded third-party content is allowed access to their first-party cookies for the duration of that page visit; on subsequent visits the embedded third-party content will get access to their first-party cookies after a user interacts with the embedded content and the content calls the Storage Access API. And, because the user previously allowed this access, they aren't re-prompted. The user's decision is maintained for the combination of first-and third-parties and is cleared when a user clears their Safari history.

Existing cookies from domains classified as having tracking abilities are purged if a user hasn't interacted with that domain—directly or through the Storage Access API—for 30 days of active Safari usage. After 30 days without interaction, a domain classified as having tracking abilities also can't set new cookies. Safari never allows access to other first-party website data in third-party contexts.

Through ITP's isolation of first-party and third-party data, it helps prevent the use of cookies and website data for the purposes of cross-site tracking. Apple has no access to which domain names a particular device has captured statistics for or classified as having tracking abilities.

In addition to blocking third-party cookies from domains classified as having tracking abilities, ITP also trims the HTTP Referer information sent to third-party domains classified as having tracking abilities to just the page's origin.

User Password Management

iOS offers a number of features to make it easy for users to securely and conveniently authenticate to third-party apps and websites that use passwords for authentication. Passwords are saved to a special Password AutoFill keychain that is user-controlled and manageable in Settings > Passwords & Accounts > Website & App Passwords. Apps can't access the Password AutoFill keychain without user permission. Credentials saved to the Password AutoFill keychain are synchronized across devices with iCloud Keychain when it is enabled.

The iCloud Keychain password manager and Password AutoFill provide the following features:

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

App access to saved passwords

Shared web credentials API

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

- `SecRequestSharedWebCredential`
- `SecAddSharedWebCredential`

Access is granted to iOS apps 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 they've been approved by Apple.

When an app with the *com.apple.developer.associated-domains* entitlement is installed, iOS makes 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 marks 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 AutoFill for Apps

iOS allows users to input saved user names and passwords into credential-related fields in apps by tapping a "key" affordance in the iOS keyboard's QuickType bar. It leverages the same app-website association mechanism powered by the apple-app-site-association file to strongly associate apps and websites. This interface exposes no credential information to the app until a user consents to release a credential to the app. When iOS has marked a website and app as having a trusted relationship, the QuickType bar will also directly suggest credentials to fill into the app. This allows users to choose to disclose Safari-saved credentials to apps with the same security properties, but without apps having to adopt an API.

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

Automatic strong passwords

When iCloud Keychain is enabled, iOS creates strong, random, unique passwords when users sign up for or change their password within an app or on a website in Safari. Users must opt-out of using strong passwords. Generated passwords are saved in the keychain and synchronized across devices with iCloud Keychain, when enabled.

Passwords generated by iOS, by default, are 20 characters long. They contain one digit, one upper-case character, two hyphens, and 16 lower-case characters. These generated passwords are strong, containing 71 bits of entropy.

iOS will generate passwords in apps and in Safari based on heuristics that determine that a password-field experience is for password creation. If the heuristics fail to recognize a password context as for password creation, app developers can set UITextContentType.newPassword on their text field, and web developers can set autocomplete="new-password" on their <input> elements.

Apps and websites can provide rules to iOS that ensure generated passwords are compatible with the relevant service. iOS will generate the strongest password it can that fulfills these rules. Developers provide these rules using UITextInputPasswordRules or the passwordrules attribute on their <input> elements.

Sending passwords to other people or devices

AirDrop

When iCloud is enabled, users can AirDrop a saved credential, including the websites its 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. (See "AirDrop security" for more information.) On the receiving device, after user consent, the credential will be stored in the user's Password AutoFill keychain.

Apple TV

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

Any iPhone nearby displays a prompt inviting the user to share a credential with the Apple TV. An iPhone and Apple TV which use the same iCloud account encrypt communication between the two devices during this process. If the iPhone is signed in to a different iCloud account than the Apple TV:

- A PIN code is used to establish an encrypted connection
- The iPhone must be unlocked and in close proximity to the Siri Remote paired to that Apple TV to receive this prompt.

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

Credential provider extensions

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

Device Controls

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

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

Passcode protection

By default, the user's passcode can be defined as a numeric PIN. On devices with Touch ID or Face ID, the minimum passcode length is six digits. On other devices, the minimum length is four digits. Users can specify a longer alphanumeric passcode by selecting Custom Alphanumeric Code in the Passcode Options in Settings > Passcode. Longer and more complex passcodes are harder to guess or attack, and are recommended.

Administrators can enforce complex passcode requirements and other policies using MDM or Exchange ActiveSync, or by requiring users to manually install configuration profiles. The following passcode policies are available:

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

For administrator details about each policy, go to:
https://help.apple.com/deployment/mdm/#/mdm4D6A472A

For developer details about each policy, go to:
https://developer.apple.com/library/ios/featuredarticles/iPhoneConfigurationProfileRef/

iOS pairing model

iOS uses 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 uses this sign of trust to enable additional functionality with the connected host, such as data synchronization.

In iOS 9, services that require pairing can’t be started until after the device has been unlocked by the user.

Additionally in iOS 10, some services, including photo syncing, require the device to be unlocked to begin.

Beginning in iOS 11, services won’t start unless the device has been recently unlocked.

The pairing process requires the user to unlock the device and accept the pairing request from the host. Starting in iOS 11, the user is also required to enter their passcode. After the user has done this, 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 (refer to “Escrow keybag” within the “Keybags” section of this paper). The exchanged keys are used to start an encrypted SSL session, which the device requires before it will send protected data to the host or start a service (iTunes syncing, file transfers, Xcode development, etc.). The device requires connections from a host over Wi-Fi to use this encrypted session for all communication, so it must have been previously paired over USB. Pairing also enables several diagnostic capabilities. In iOS 9, if a pairing record hasn’t been used for more than six months, it expires. This timeframe is shortened to 30 days in iOS 11.

For more information, go to:
https://support.apple.com/kb/HT6331

Certain services, including com.apple.pcapd, are restricted to work only over USB. Additionally, the com.apple.file_relay service requires an Apple-signed configuration profile to be installed.

In iOS 11, 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.

For more information, go to:
https://support.apple.com/kb/HT5868

Configuration enforcement

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

An iOS configuration profile contains a number of settings that can be specified, including:

- Passcode policies
- Restrictions on device features (disabling the camera, for example)
- Wi-Fi settings
- VPN settings

- Mail server settings
- Exchange settings
- LDAP directory service settings
- CalDAV calendar service settings
- Web clips
- Credentials and keys
- Advanced cellular network settings

To view a current list for administrators, go to:
https://help.apple.com/deployment/mdm/#/mdm5370d089

To view a current list for developers, go to:
https://developer.apple.com/library/ios/featuredarticles/iPhoneConfigurationProfileRef/

Configuration profiles can be signed and encrypted to validate their origin, ensure their integrity, and protect their contents. Configuration profiles are encrypted using CMS (RFC 3852), supporting 3DES and AES-128.

Configuration profiles can also be locked to a device to completely prevent their removal, or to allow removal only with a passcode. Since many enterprise users own their iOS 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.

Users can install configuration profiles directly on their devices using Apple Configurator 2, or they can be downloaded via Safari, sent via a mail message, or sent over the air using an 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.

Mobile device management (MDM)

iOS support for MDM allows businesses to securely configure and manage scaled iPhone, iPad, Apple TV, and Mac deployments across their organizations. MDM capabilities are built on existing iOS technologies such as configuration profiles, over-the-air enrollment, and the Apple Push Notification service. For example, APNs is used to wake the device so it can communicate directly with its MDM solution over a secured connection. No confidential or proprietary information is transmitted via APNs.

Using MDM, IT departments can enroll iOS 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.

For more information on MDM, go to:

- https://www.apple.com/iphone/business/it/management.html
- https://help.apple.com/deployment/ios/#/ior07301dd60
- https://help.apple.com/deployment/mdm/#/mdmbf9e668

Shared iPad

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

Signing in to Shared iPad

When a student signs in, the Managed Apple ID is authenticated with Apple's identity servers using the SRP protocol. If successful, a short-lived access token specific to the device is granted. If the student has used the device before, they already have a local user account that is unlocked using the same credential.

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

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

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

Signing out of Shared iPad

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

Shared iPad upgrades

When a Shared iPad is upgraded from a version prior to iOS 10.3 to a version of 10.3 or later, a one-time file system conversion takes place to convert the HFS+ data partition to an APFS volume. If, at that time, any user home directories are present on the system, they will remain on the main data volume instead of being converted to individual APFS volumes.

When additional students sign in, their home directories will also be placed on the main data volume. New user accounts won't be created with their own APFS volume, as described previously, until all user accounts on the main data volume have been deleted. Thus, to ensure that users have the additional protections and quotas afforded by APFS, the iPad should either be upgraded to 10.3 or later via an erase-and-re-install, or all user accounts on the device should be deleted through the Delete User MDM command.

For more information on Shared iPad, go to:
https://help.apple.com/deployment/mdm/#/cad7e2e0cf56

Apple School Manager

Apple School Manager is a service for educational institutions that enables them to buy content, configure automatic device enrollment in MDM solutions, create accounts for students and staff, and set up iTunes U courses. Apple School Manager is accessible on the web and is designed for technology managers and IT administrators, staff, and teachers.

For more information on Apple School Manager, go to:
https://help.apple.com/schoolmanager/

Apple Business Manager

Apple Business Manager is a simple, web-based portal for IT administrators to deploy iOS, macOS, and tvOS devices all from one place. When used with your mobile device management (MDM) solution, you can configure device settings and buy and distribute apps and books. Apple Business Manager is accessible on the web and is designed for IT administrators.

For more information on Apple Business Manager, go to:
https://help.apple.com/businessmanager/

Device Enrollment

Apple School Manager and Apple Business Manager provide a fast, streamlined way to deploy iOS devices that an organization has purchased directly from Apple or through participating Apple Authorized Resellers and carriers. Devices running 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.

Organizations can automatically enroll devices in MDM without having to physically touch or prep the devices before users get them. After enrolling in one of the programs, administrators sign in to the program website and link the program to their MDM solution. The devices they purchased can then be assigned to users through MDM. 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 (SSL).

The setup process for users can be further simplified by removing specific steps in the Setup Assistant for iOS, tvOS, and macOS, so users are up and running quickly. Administrators can also control whether or not the user can remove the MDM profile from the device and ensure that device restrictions are in place from the very start. 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.

Apple Configurator 2

In addition to MDM, Apple Configurator 2 for macOS makes it easy to set up and preconfigure iOS devices and Apple TV before handing them out to users. With Apple Configurator 2, devices can be quickly preconfigured with apps, data, restrictions, and settings.

Apple Configurator 2 allows you to use Apple School Manager or Apple Business Manager to enroll devices in an MDM solution without users having to use the Setup Assistant. Apple Configurator 2 can also be used to add iOS devices and Apple TV to Apple School Manager or Apple Business Manager after the time of purchase.

For more information on Apple Configurator 2, go to:
https://help.apple.com/configurator/mac/

Supervision

During the setup of a device, an organization can configure the device to be supervised. Supervision denotes that the device is institutionally owned, which provides additional control over its configuration and restrictions. With Apple School Manager or Apple Business Manager, supervision can be wirelessly enabled on the device as part of the MDM enrollment process, or enabled manually using Apple Configurator 2. Supervising a device requires the device to be erased and the operating system reinstalled.

For more information on configuring and managing iOS devices and Apple TV using MDM or Apple Configurator 2, go to:
https://help.apple.com/deployment/ios/

Restrictions

Restrictions can be enabled—or in some cases, disabled—by administrators to prevent users from accessing a specific app, service, or function of the device. Restrictions are sent to devices in a restrictions payload, which is attached to a configuration profile. Restrictions can be applied to iOS, tvOS, and macOS devices. Certain restrictions on a managed iPhone may be mirrored on a paired Apple Watch.

To view a current list for IT managers, go to:
https://help.apple.com/deployment/mdm/#/mdm0F7DD3D8

Remote wipe

iOS devices can be erased remotely by an administrator or user. Instant remote wipe is achieved by securely discarding the block storage encryption key from Effaceable Storage, rendering all data unreadable. A remote wipe command can be initiated by MDM, Exchange, or iCloud.

When a remote wipe command is triggered by MDM or iCloud, the device sends an acknowledgment and performs the wipe. For remote wipe through Exchange, the device checks in with the Exchange server before performing the wipe.

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

Lost Mode

If a device is lost or stolen, an MDM administrator can remotely enable Lost Mode on a supervised device with iOS 9.3 or later. When Lost Mode is enabled, the current user is logged out and the device can't be unlocked. The screen displays a message that can be customized by the administrator, such as displaying a phone number to call if the device is found. When the device is put into Lost Mode, the administrator can request the device to send its current location and, optionally, play a sound. When an administrator turns off Lost Mode, which is the only way the mode can be exited, the user is informed of this action through a message on the Lock screen or an alert on the Home screen.

Activation Lock

When Find My iPhone is turned on, the device can't be reactivated without entering the owner's Apple ID credentials or the previous passcode of the device.

With devices that are owned by an organization, it's a good idea to supervise devices so that Activation Lock can be managed by the organization instead of relying on the individual user to enter their Apple ID credentials to reactivate devices.

On supervised devices, a compatible MDM solution can store a bypass code when Activation Lock is enabled, or later use this code to clear Activation Lock automatically when the device needs to be erased and assigned to a new user.

By default, supervised devices never have Activation Lock enabled, even if the user turns on Find My iPhone. However, an MDM solution may retrieve a bypass code and permit Activation Lock to be enabled on the device. If Find My iPhone is turned on when the MDM solution enables Activation Lock, it is enabled at that point. If Find My iPhone is turned off when the MDM server enables Activation Lock, it's enabled the next time the user activates Find My iPhone.

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

Screen Time

Screen Time is a feature in iOS 12 that lets a user understand and control their own app and web usage, or that of their children. Users can:

- View usage data
- Set app or web usage limits
- Configure Downtime
- Enforce additional restrictions

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

When a user clears Safari history or deletes an app, the corresponding usage data is removed from the device and all synchronized devices.

Parents and Screen Time

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

Usage data and configuration settings are transferred between the parent's and child's devices using an end-to-end encrypted connection to Apple identity service (IDS). Encrypted data may be briefly stored on IDS servers until it is read by the receiving device (for example, as soon as the iPhone/iPad 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 Apple can better understand how Screen Time is being used:

- Was Screen Time turned on during Setup Assistant or later in Settings
- Is Screen Time turned on
- Is Downtime enabled
- Number of times the "Ask for more" query was used
- Number of app limits

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.

Privacy Controls

Apple takes customer privacy seriously and has numerous built-in controls and options that allow iOS users to decide how and when apps utilize their information, as well as what information is being utilized.

Location Services

Location Services uses GPS, Bluetooth, and crowd-sourced Wi-Fi hotspot and cell tower locations to determine the user's approximate location. Location Services can be turned off using a single switch in Settings, or users can approve access for each app that uses the service. Apps may request to receive location data only while the app is being used or allow it at any time. Users may choose not to allow this access, and may change their choice at any time in Settings. From Settings, access can be set to never allowed, allowed when in use, or always, depending on the app's requested location use. Also, if apps granted access to use location at any time make use of this permission while in background mode, users are reminded of their approval and may change an app's access.

Additionally, users are given fine-grained control over system services' use of location information. This includes being able to turn off the inclusion of location information in information collected by the analytics services used by Apple to improve iOS, location-based Siri information, location-based context for Siri Suggestions searches, local traffic conditions, and significant locations visited in the past.

Access to personal data

iOS helps prevent apps from accessing a user's personal information without permission. Additionally, in Settings, users can see which apps they have permitted to access certain information, as well as grant or revoke any future access. This includes access to:

- Contacts
- Calendars
- Reminders
- Photos
- Motion activity and fitness
- Location Services
- Apple Music
- Your music and video activity
- Microphone
- Camera
- HomeKit
- Health
- Speech recognition
- Bluetooth sharing
- Your media library

If the user signs in to iCloud, apps are granted access by default to iCloud Drive. Users may control each app's access under iCloud in Settings. Additionally, iOS provides restrictions that prevent data movement between apps and accounts installed by an MDM solution and those installed by the user.

Privacy policy

To read Apple's privacy policy, go to:
https://www.apple.com/legal/privacy

Security Certifications and Programs

Note: For the latest information on iOS Security Certifications, validations, and guidance, go to:
https://support.apple.com/kb/HT202739

ISO 27001 and 27018 certifications

Apple has received ISO 27001 and ISO 27018 certifications for the Information Security Management System for the infrastructure, development, and operations supporting these products and services: Apple School Manager, iTunes U, iCloud, iMessage, FaceTime, Managed Apple IDs, Siri, and Schoolwork in accordance with the Statement of Applicability v2.1 dated 7/11/2017. Apple's compliance with the ISO standards was certified by the British Standards Institution. The BSI website has certificates of compliance for ISO 27001 and ISO 27018. To view these certificates, go to:

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

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

Cryptographic validation (FIPS 140-2)

The cryptographic modules in iOS have been repeatedly validated for compliance with U.S. Federal Information Processing Standards (FIPS) 140-2 following each release since iOS 6. As with each major release, Apple submits the modules to CMVP for re-validation when the iOS operating system is released. This program validates the integrity of cryptographic operations for Apple apps and third-party apps that properly utilize iOS cryptographic services and approved algorithms.

Apple received FIPS 140-2 Validation for the embedded hardware module identified as **Apple Secure Enclave Processor (SEP) Secure Key Store (SKS) Cryptographic Module** enabling approved use of SEP generated and managed keys. Apple will continue to pursue higher levels for the hardware module with each successive major iOS release as appropriate.

Common Criteria Certification (ISO 15408)

Since the release of iOS 9, Apple has achieved ISO certifications for each major iOS release under the Common Criteria Certification program and has expanded coverage to include the following:

- Mobile Device Fundamental Protection Profile
  - Extended Package for Mobile Device Management Agents
  - Extended Package for Wireless LAN Clients
  - PP-Module for VPN Client
- Protection Profile for Application Software
  - Extended Package for Web Browsers

iOS 12 is expected to include additional certifications for the following:

- Extended Package for Email Clients

Apple plans to expand coverage with each successive major release of iOS.

Apple has taken an active role within the International Technical Community (ITC) in developing currently unavailable Collaborative Protection Profiles (cPPs) focused on evaluating key mobile security technology. Apple continues to evaluate and pursue certifications against new and updated versions of the cPPs available today and under development.

Commercial Solutions for Classified (CSfC)

Where applicable, Apple has also submitted the iOS platform and various services for inclusion in the Commercial Solutions for Classified (CSfC) Program Components List. As Apple platforms and services undergo Common Criteria Certifications, they will be submitted for inclusion under CSfC Program Components List as well.

To view the most recently listed components, go to:
https://www.nsa.gov/resources/everyone/csfc/components-list/

Security configuration guides

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

Apple Security Bounty

Apple rewards researchers who share critical issues with Apple. In order to be eligible for an Apple Security Bounty, researchers are required to provide a clear report and working proof of concept. The vulnerability must affect the latest shipping iOS and, where relevant, the latest hardware. The exact payment amount will be determined after review by Apple. The criteria includes novelty, likelihood of exposure, and degree of user interaction required.

Once the issues are properly shared, Apple makes it a priority to resolve confirmed issues as quickly as possible. Where appropriate, Apple will provide public recognition, unless otherwise requested.

Category	**Maximum payment (USD)**
Secure boot firmware components	$200,000
Extraction of confidential material protected by the Secure Enclave	$100,000
Execution of arbitrary code with kernel privileges	$50,000
Unauthorized access to iCloud account data on Apple servers	$50,000
Access from a sandboxed process to user data outside of that sandbox	$25,000

Conclusion

A commitment to security

Apple is committed to helping protect customers with leading privacy and security technologies that are designed to safeguard personal information, as well as comprehensive methods to help protect corporate data in an enterprise environment.

Security is built into iOS. From the platform to the network to the apps, everything a business needs is available in the iOS platform. Together, these components give iOS its industry-leading security without compromising the user experience.

Apple uses a consistent, integrated security infrastructure throughout iOS and the iOS apps ecosystem. Hardware-based storage encryption provides remote wipe capabilities when a device is lost, and enables users to completely remove all corporate and personal information when a device is sold or transferred to another owner. Diagnostic information is also collected anonymously.

iOS apps designed by Apple are built with enhanced security in mind. For example, iMessage and FaceTime provide client-to-client encryption. For third-party apps, the combination of required code signing, sandboxing, and entitlements gives users industry-leading protection against viruses, malware, and other exploits. The App Store submission process works to further shield users from these risks by reviewing every iOS app before it’s made available.

To make the most of the extensive security features built into iOS, businesses 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 this platform.

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

To learn more about reporting issues to Apple and subscribing to security notifications, go to:
https://www.apple.com/support/security

Glossary

Address space layout randomization (ASLR)	A technique employed by iOS to make the successful exploitation by a software bug much more difficult. By ensuring memory addresses and offsets are unpredictable, exploit code can't hard code these values. In iOS 5 or later, the position of all system apps and libraries are randomized, along with all third-party apps compiled as position-independent executables.
Apple identity service (IDS)	Apple's directory of iMessage public keys, APNs addresses, and phone numbers and email addresses that are used to look up the keys and device addresses.
Apple Push Notification service (APNs)	A worldwide service provided by Apple that delivers push notifications to iOS devices.
Boot Progress Register (BPR)	A set of SoC hardware flags that software can use to track the boot modes the device has entered, such as DFU Mode and Recovery Mode. Once a Boot Progress Register flag is set, it can't be cleared. This allows later software to get a trusted indicator of the state of the system.
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.
Data Protection	File and Keychain protection mechanism for iOS. It can also refer to the APIs that apps use to protect files and Keychain items.
Device Firmware Upgrade (DFU)	A mode in which a device's Boot ROM code waits to be recovered over USB. The screen is black when in DFU mode, but upon connecting to a computer running iTunes, the following prompt is presented: "iTunes has detected an iPad in recovery mode. You must restore this iPad before it can be used with iTunes."
Elliptic Curve Diffie-Hellman Exchange (ECDHE)	Elliptic Curve Diffie-Hellman Exchange with ephemeral keys. ECDHE allows two parties to agree on a secret key in a way that prevents the key from being discovered by an eavesdropper watching the messages between the two parties.
ECDSA	A digital signature algorithm based on elliptic curve cryptography.
Exclusive Chip Identification (ECID)	A 64-bit identifier that's unique to the processor in each iOS device. When a call is answered on one device, ringing of nearby iCloud-paired devices is terminated by briefly advertising through Bluetooth Low Energy 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.
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.
file system key	The key that encrypts each file's metadata, including its class key. This is kept in Effaceable Storage to facilitate fast wipe, rather than confidentiality.
group ID (GID)	Like the UID, but common to every processor in a class.
hardware security module (HSM)	A specialized tamper-resistant computer that safeguards and manages digital keys.
iBoot	Code that loads XNU, as part of the secure boot chain. Depending on the SoC generation, iBoot may be loaded by LLB or directly by the boot ROM.
integrated circuit (IC)	Also known as a microchip.
Joint Test Action Group (JTAG)	Standard hardware debugging tool used by programmers and circuit developers.

keybag	A data structure used to store a collection of class keys. Each type (user, device, system, backup, escrow, or iCloud Backup) has the same format: • A header containing: – Version (set to three in iOS 5) – Type (system, backup, escrow, or iCloud Backup) – Keybag UUID – An HMAC if the keybag is signed – 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 this is) – Wrapping type (UID-derived key only; UID-derived key and passcode-derived key) – Wrapped class key – Public key for asymmetric classes
Keychain	The infrastructure and a set of APIs used by iOS and third-party apps to store and retrieve passwords, keys, and other sensitive credentials.
key wrapping	Encrypting one key with another. iOS uses NIST AES key wrapping, as per RFC 3394.
Low-Level Bootloader (LLB)	On systems with a two-stage boot architecture, code that's invoked by the Boot ROM, and in turn loads iBoot, as part of the secure boot chain.
memory controller	The subsystem in the SoC that controls the interface between the SoC and its main memory.
per-file key	The AES-256-bit key used 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 plist signed by Apple that contains a set of entities and entitlements allowing apps to be installed and tested on an iOS device. A development Provisioning Profile lists the devices that a developer has chosen for ad hoc distribution, and a distribution Provisioning Profile contains the app ID of an enterprise-developed app.
ridge flow angle mapping	A mathematical representation of the direction and width of the ridges extracted from a portion of a fingerprint.
software seed bits	Dedicated bits in the Secure Enclave AES engine that get appended to the UID when generating keys from the UID. Each software seed bit has a corresponding lock bit. The Secure Enclave Boot ROM and OS can independently change the value of each software seed bit as long as the corresponding lock bit hasn't been set. Once the lock bit is set, neither the software seed bit nor the lock bit can be modified. The software seed bits and their locks are reset when the Secure Enclave reboots.
System Coprocessor Integrity Protection (SCIP)	System coprocessors are CPUs on the same SoC as the application processor.
system on chip (SoC)	An integrated circuit (IC) that incorporates multiple components into a single chip. The application processor, Secure Enclave and other coprocessors are components of the SoC.
tangling	The process by which a user's passcode is turned into a cryptographic key and strengthened with the device's UID. This ensures that a brute-force attack must be performed on a given device, and thus is rate limited and can't be performed in parallel. The tangling algorithm is PBKDF2, which uses AES keyed with the device UID as the pseudorandom function (PRF) for each iteration.
Uniform Resource Identifier (URI)	A string of characters that identifies a web-based resource.
Unique ID (UID)	A 256-bit AES key that's burned into each processor at manufacture. It can't be read by firmware or software, and is used only by the processor's hardware AES engine. To obtain the actual key, an attacker would have to mount a highly sophisticated and expensive physical attack against the processor's silicon. The UID isn't related to any other identifier on the device including, but not limited to, the UDID.
XNU	The kernel at the heart of the iOS and macOS operating systems. It's assumed to be trusted, and enforces security measures such as code signing, sandboxing, entitlement checking, and ASLR.

Document Revision History

Date	Summary
September 2018	**Updated for iOS 12** • Secure Enclave • OS Integrity Protection • Express Card with power reserve • DFU 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
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 **Updated for iOS 11.1** • Security Certifications and programs • Touch ID/Face ID • Shared Notes • CloudKit end-to-end encryption • TLS • Apple Pay, Paying with Apple Pay on the web • Siri Suggestions • Shared iPad For more information about the security contents of iOS 11 go to: https://support.apple.com/HT208112

Date	Summary
July 2017	**Updated for iOS 10.3** • System Enclave • File Data Protection • Keybags • Security Certifications and programs • SiriKit • HealthKit • Network Security • Bluetooth • Shared iPad • Lost Mode • Activation Lock • Privacy Controls For more information about the security contents of iOS 10.3 go to: https://support.apple.com/HT207617
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 For more information about the security contents of iOS 10 go to: https://support.apple.com/HT207143
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 For more information about the security contents of iOS 9.3 go to: https://support.apple.com/HT206166

Date	Summary
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 For more information about the security contents of iOS 9 go to: https://support.apple.com/HT205212