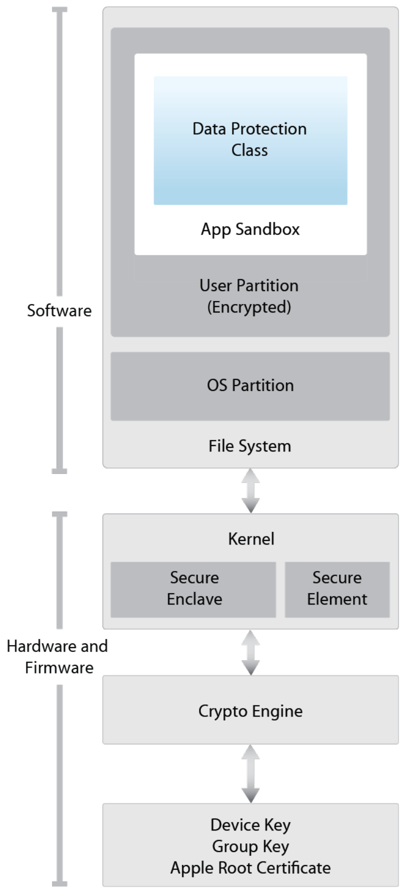
iOS Security

iOS 9.0 or later

September 2015

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 are not configurable, so users can't disable them by mistake. Other features, such as Touch 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.
- **Device controls:** Methods that prevent unauthorized use of the device and enable it to be remotely wiped if lost or stolen.
- **Privacy controls:** Capabilities of iOS that can be used to control access to Location Services and user data.

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 and software 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.

When an iOS device is turned on, its application processor immediately executes code from read-only memory known as the 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 Low-Level Bootloader (LLB) 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 LLB finishes its tasks, it verifies and runs the next-stage bootloader, iBoot, which in turn verifies and runs the iOS kernel.

This secure boot chain helps ensure that the lowest levels of software are not tampered with and allows iOS to run only on validated Apple devices.

For 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.

For devices with an A7 or later A-series processor, the Secure Enclave coprocessor also utilizes a secure boot process that ensures its separate software is verified and signed by Apple.

If one step of this boot process is unable to load or verify the next process, startup is stopped and the device displays the "Connect to iTunes" screen. This is called recovery mode. If the Boot ROM is not able to load or verify LLB, it enters DFU (Device Firmware Upgrade) mode. In both cases, the device must be connected to iTunes via USB and restored to factory default settings. For more information on manually entering recovery mode, see https://support.apple.com/kb/HT1808.

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 hold down both the Home and Sleep/Wake buttons. After 8 seconds, release the Sleep/Wake button while continuing to hold down the Home button. Note: Nothing will be displayed on the screen when the device is in DFU mode. If the Apple logo appears, the Sleep/Wake button was held down too long.

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 above 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 an A7 or later A-series processor, the Secure Enclave coprocessor also utilizes System Software Authorization to ensure the integrity of its software and prevent downgrade installations. See "Secure Enclave," below.

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 local network server running the caching service on OS X Server so that iOS devices do not need to access Apple servers to obtain the necessary update data.

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, LLB, iBoot, the kernel, and OS image), a random anti-replay value (nonce), and the device's unique ID (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 in the Apple A7 or later A-series processor. It utilizes its own secure boot and personalized software update separate from the application processor. It provides all cryptographic operations for Data Protection key management and maintains the integrity of Data Protection even if the kernel has been compromised.

The Secure Enclave uses encrypted memory and includes a hardware random number generator. Its microkernel is based on the L4 family, with modifications by Apple. Communication between the Secure Enclave and the application processor is isolated to an interrupt-driven mailbox and shared memory data buffers.

Each Secure Enclave is provisioned during fabrication with its own UID (Unique ID) that is not accessible to other parts of the system and is not known to Apple. When the device starts up, an ephemeral key is created, entangled with its UID, and used to encrypt the Secure Enclave's portion of the device's memory space.

Additionally, data that is saved to the file system by the Secure Enclave is encrypted with a key entangled with the UID and an anti-replay counter.

The Secure Enclave is responsible for processing fingerprint data from the Touch ID sensor, determining if there is a match against registered fingerprints, and then enabling access or purchases on behalf of the user. 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 cannot read it. It's encrypted and authenticated with a session key that is negotiated using the device's shared key that is provisioned for the Touch ID sensor and the Secure Enclave. 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.

Touch ID

Touch ID is the fingerprint sensing system that makes secure access to the device 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.

Touch ID makes using a longer, more complex passcode far more practical because users won't have to enter it as frequently. Touch ID also overcomes the inconvenience of a passcode-based lock, not by replacing it but by securely providing access to the device within thoughtful boundaries and time constraints.

Touch ID and passcodes

To use Touch ID, users must set up their device so that a passcode is required to unlock it. When Touch ID scans and recognizes an enrolled fingerprint, the device unlocks without asking for the device passcode. The passcode can always be used instead of Touch ID, and it's still required under the following circumstances:

- The device has just been turned on or restarted.
- The device has not been unlocked for more than 48 hours.
- The device has received a remote lock command.
- After five unsuccessful attempts to match a fingerprint.
- When setting up or enrolling new fingers with Touch ID.

When Touch ID is enabled, the device immediately locks when the Sleep/Wake button is pressed. With passcode-only security, many users set an unlocking grace period to avoid having to enter a passcode each time the device is used. With Touch ID, the device locks every time it goes to sleep, and requires a fingerprint—or optionally the passcode—at every wake.

Touch ID can be trained to recognize up to five different fingers. With one finger enrolled, the chance of a random match with someone else is 1 in 50,000. However, Touch ID allows only five unsuccessful fingerprint match attempts before the user is required to enter a passcode to obtain access.

Other uses for Touch ID

Touch ID can also be configured to approve purchases from the iTunes Store, the App Store, and the iBooks Store, so users don't have to enter an Apple ID password. When they choose to authorize a purchase, authentication tokens are exchanged between the device and the store. The token and cryptographic nonce are held in the Secure Enclave. The nonce is signed with a Secure Enclave key shared by all devices and the iTunes Store.

Touch ID can also be used with Apple Pay, Apple's implementation of secure payments. For more information, see the Apple Pay section of this document.

Additionally, third-party apps can use system-provided APIs to ask the user to authenticate using Touch ID or a passcode. The app is only notified as to whether the authentication was successful; it cannot access Touch ID or the data associated with the enrolled fingerprint.

Keychain items can also be protected with Touch ID, to be released by the Secured Enclave only by a fingerprint match or the device passcode. App developers also have APIs to verify that a passcode has been set by the user and therefore able to authenticate or unlock keychain items using Touch ID.

With iOS 9, developers can require that Touch ID API operations don't fall back to an application password or the device passcode. Along with the ability to retrieve a representation of the state of enrolled fingers, this allows Touch ID to be used as a second factor in security sensitive apps.

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.

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 subdermal 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, and is never sent to Apple or backed up to iCloud or iTunes.

How Touch ID unlocks an iOS device

If Touch ID is turned off, when a device locks, the keys for Data Protection class Complete, which are held in the Secure Enclave, are discarded. The files and keychain items in that class are inaccessible until the user unlocks the device by entering his or her passcode.

With Touch ID turned on, the keys are not discarded when the device locks; instead, they're wrapped with a key that is given to the Touch ID subsystem inside the Secure Enclave. When a user attempts to unlock the device, if Touch ID recognizes the user's fingerprint, it provides the key for unwrapping the Data Protection keys, and the device is unlocked. This process provides additional protection by requiring the Data Protection and Touch ID subsystems to cooperate in order to unlock the device.

The keys needed for Touch ID to unlock the device are lost if the device reboots and are discarded by the Secure Enclave after 48 hours or five failed Touch ID recognition attempts.

Encryption and Data Protection

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.

The device's unique ID (UID) and a device group ID (GID) 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. Additionally, the Secure Enclave's UID and GID can only be used by the AES engine dedicated to the Secure Enclave. The UIDs are unique to each device and are not recorded by Apple or any of its suppliers. The GIDs are common to all processors in a class of devices (for example, all devices using the Apple A8 processor), and are used for non security-critical tasks such as when delivering system software during installation and restore. Integrating these keys into the silicon helps prevent them from being tampered with or bypassed, or accessed outside the AES engine. The UIDs and GIDs are also not available via JTAG or other debugging interfaces.

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 is not related to any other identifier on the device.

Erase all content and settings

The "Erase all content and settings" option in Settings obliterates all 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: Do not use the "Erase all content and settings" option until the device has been backed up, as there is no way to recover the erased data.

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. System entropy is generated from timing variations during boot, and additionally from interrupt timing once 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, 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.

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.

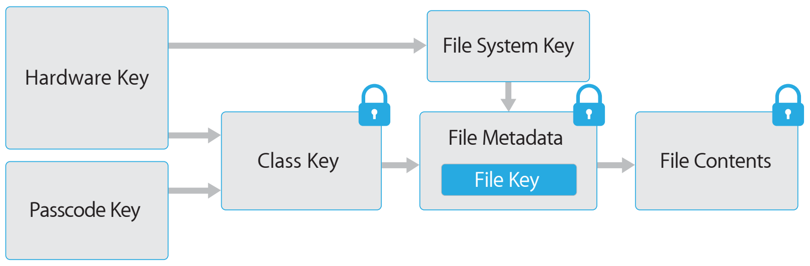
Architecture overview

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 CBC mode. (On devices with an A8 processor, AES-XTS is used.) The initialization vector (IV) is calculated with the block offset into the file, encrypted with the SHA-1 hash of the per-file key.

The per-file 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.

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 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, 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. The file system key is stored in Effaceable Storage. Since it's stored on the device, this key is not used to maintain the confidentiality of data; instead, it's designed to be quickly erased on demand (by the user, with the "Erase all content and settings" option, or by a user or administrator issuing a remote wipe command from a mobile device management (MDM) server, Exchange ActiveSync, or iCloud). Erasing the key in this manner renders all files cryptographically inaccessible.



The content of a file is encrypted with a per-file key, which is wrapped with a class key and stored in a file's metadata, which is in turn 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.

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.

Passcodes

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 5½ years to try all combinations of a six-character alphanumeric passcode with lowercase letters and numbers.

The stronger the user passcode is, the stronger the encryption key becomes. Touch ID 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.

Delays between passcode attempts

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

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. This setting is also available as an administrative policy through mobile device management (MDM) and Exchange ActiveSync, and can be set to a lower threshold.

On devices with an A7 or later A-series processor, the delays are enforced by the Secure Enclave. If the device is restarted during a timed delay, the delay is still enforced, with the timer starting over for the current period.

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.

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; its hash is used to unwrap the per-file key, which 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 is not 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 is not assigned a Data Protection class, it is still stored in encrypted form (as is all data on an iOS device).

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.

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

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

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.

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 only available when the device is configured with a passcode. This class exists only in the system keybag; they do not sync to iCloud Keychain, are not backed up, and are not 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 token	Always, non-migratory
iCloud certificates and private key	Always, non-migratory
iMessage keys	Always, non-migratory
Certificates and private keys installed by 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 or by entering the device's passcode. ACLs are evaluated inside the Secure Enclave and are released to the kernel only if their specified constraints are met.

Access to Safari saved passwords

iOS apps can interact with keychain items saved by Safari for password autofill using the following two APIs:

- `SecRequestSharedWebCredential`
- `SecAddSharedWebCredential`

Access will be granted only if both 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. The websites must place a file on their server listing the unique app identifiers of apps they've approved. When an app with the com.apple.developer.associated-domains entitlement is installed, iOS makes a TLS request to each listed website, requesting the file/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, or are updated or deleted.

Keybags

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

System 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 system keybag and unwrapped. It is a binary plist stored in the No Protection class, but 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 system 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 system keybag are accessible, and have been unwrapped successfully.

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 earlier, 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 10,000 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 are not 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 MDM. This keybag allows iTunes to back up and sync without requiring the user to enter a passcode, and it allows an MDM server to remotely clear a user's passcode. It is stored on the computer that's used to sync with iTunes, or on the MDM server that 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 OTA software update, the user is prompted for his or her passcode when initiating the update. This is used to securely create a One-time Unlock Token, which unlocks the system keybag after the update. This token cannot 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 SEP 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 has not rebooted within 20 minutes.

For unattended software updates, which is set when the user chooses "Install Later" when notified of the update, the application processor can keep the One-time Unlock Token alive in the Secure Enclave for up to 8 hours. After that time, a policy timer increments the counter.

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.

Security Certifications and programs

Cryptographic Validation (FIPS 140-2)

The cryptographic modules in iOS have been validated for compliance with U.S. Federal Information Processing Standards (FIPS) 140-2 Level 1 following each releases since iOS 6. The cryptographic modules in iOS 9 are identical to those in iOS 8, but as with each release, Apple submits the modules for re-validation. This program validates the integrity of cryptographic operations for Apple apps and third-party apps that properly utilize iOS cryptographic services.

Common Criteria Certification (ISO 15408)

Apple has already begun pursuit of iOS certification under the Common Criteria Certification (CCC) program. The first two certifications currently active are against the Mobile Device Fundamental Protection Profile v2.0 (MDFPP2) and the VPN IPSecPP1.4 Client Protection Profile (VPNIPSecPP1.4). Apple has taken an active role within the International Technical Community (ITC) in developing currently unavailable Protection Profiles (PPs) focused on evaluating key mobile security technology. Apple continues to evaluate and pursue certifications against new and updated version of the PPs available today.

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. Specifically, iOS for Mobile Platform and the IKEv2 client for the IPSec VPN Client (IKEv2 Always-On VPN only). As Apple platforms and services undergo Common Criteria Certifications, they will be submitted for inclusion under CSfC Program Component List as well.

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." These guides provide defined and vetted information about how to configure and utilize features in iOS for enhanced protection.

For information on iOS security certifications, validations, and guidance, see https://support.apple.com/kb/HT202739.

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

Once 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 have not 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 iOS 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. Once 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 in order 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 via 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 does not 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 has not been modified since it was installed or last updated.

Runtime process security

Once 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 do not 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 cannot 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 application 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 applications. 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 do not 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 extensions since they are enabled by the user for the entire system. Once enabled, the extension will be used for any text field except the passcode input and any secure text view. For privacy reasons, 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 mobile device management, document and keyboard extensions obey Managed Open In rules. For example, the MDM server 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-disk container for storage, which will stay 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 (including attachments), managed books, Safari bookmarks, app launch images, and location data are also stored encrypted with keys protected by the user's passcode on their device. Calendar (excluding attachments), Contacts, Reminders, Notes, Messages, and Photos implement Protected Until First User Authentication.

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

Accessories

The Made for iPhone, iPod touch, and iPad (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 via 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 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 devices are granted full access to the device. If an accessory does not provide 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 his or her 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.

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 8-digit code provided by the accessory's manufacturer and 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.

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.

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 do not 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 is not 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, every further communication is authenticated and encrypted using Station-to-Station protocol and per-session keys.

Only the user who initially created the home in HomeKit can add new users. His or her 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. The process for configuring Apple TV for use with HomeKit uses the same authentication and encryption as when adding additional users, but is performed automatically if the user who created the home is signed in to iCloud on the Apple TV, and the Apple TV is in the home.

If a user does not have multiple devices, and does not grant additional users access to his or her 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.

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, as described in the Siri section of this paper, to provide names of rooms, accessories, and scenes that are necessary for command recognition.

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 does not 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 is 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 does not 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 is not 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 is not 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 is not authenticated, in order to help protect privacy.

Accessories connect to the iCloud remote access server using HTTP/2, secured using TLS 1.2 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.

HealthKit

The HealthKit framework provides a common database that apps can use to store and access fitness and health data with permission of the user. HealthKit also works directly with health and fitness devices, such as compatible Bluetooth LE heart rate monitors and the motion coprocessor built into many iOS devices.

Health data

HealthKit uses a database to store the user's health data, such as height, weight, distance walked, blood pressure, and so on. This database is stored in Data Protection class Complete Protection, which means it is accessible only after a user enters his or her passcode or uses Touch ID to unlock the device.

Another database stores operational data, such as access tables for apps, names of devices connected to HealthKit, and scheduling information used to launch apps when new data is available. This database 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, they are imported into the primary health databases, then deleted when the merge is completed.

Health data is not shared via iCloud or synced between devices. Health databases are included in encrypted device backups to iCloud or iTunes. Health data is not included in unencrypted iTunes backups.

Data integrity

Data stored in the database includes metadata to track the provenance of each data record. This metadata includes an application 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 are not 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 does not 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 is not 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.

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 BTLE 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 BTLE 4.1 out-of-band pairing. Standard BTLE Passkey Entry is used as a fallback pairing method, if necessary.

Once the BTLE session is established, Apple Watch and iPhone exchange keys using a process adapted from IDS, as described in the iMessage section of this paper. Once 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 BTLE and Wi-Fi links providing a secondary encryption layer. Key rolling is utilized at 15-minute intervals to limit the exposure window, should traffic be compromised.

To support apps that need streaming data, encryption is provided using methods described in the FaceTime section of this paper, utilizing the IDS service provided by the paired iPhone.

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

When Apple Watch is not within Bluetooth range, Wi-Fi can be used instead. Apple Watch will not join Wi-Fi networks unless the credentials to do so are present on the paired iPhone, which provides the list of known networks to the watch automatically.

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 locked, Apple Pay can't be used. If the automatic locking provided by wrist detection is turned off in settings, Apple Pay is disabled. Wrist detection is turned off using the Apple Watch app on iPhone. This setting can also be enforced using mobile device management.

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 is not 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. Pairing with a new iPhone automatically erases all content and data from Apple Watch.

Enabling Find My Phone on the paired iPhone also enables 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. Safari, Calendar, Mail, and other Internet apps automatically use these mechanisms 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 (SecureTransport) provide fine-grained control. By default, CFNetwork disallows SSLv3, and apps that use WebKit (such as Safari) are prohibited from making an SSLv3 connection.

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.

Servers must support a minimum of TLS 1.2, forward secrecy, and certificates must be valid and signed using SHA-256 or better with a minimum of a 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.

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.

- Pulse Secure, Cisco, Aruba Networks, SonicWALL, Check Point, Palo Alto Networks, Open VPN, AirWatch, MobileIron, NetMotion Wireless, and F5 Networks SSL-VPN using the appropriate client app from the App Store.
- Cisco IPSec with user authentication by Password, RSA SecurID or CRYPTOCard, and machine authentication by shared secret and certificates.
- L2TP/IPSec with user authentication by MS-CHAPV2 Password, RSA SecurID or CRYPTOCard, and machine authentication by shared secret.
- PPTP with user authentication by MS-CHAPV2 Password and RSA SecurID or CRYPTOCard is supported, but not recommended.

iOS supports VPN On Demand for networks that use certificate-based authentication. IT policies specify which domains require a VPN connection by using a configuration profile.

iOS also supports Per App VPN support, facilitating VPN connections on a much more granular basis. Mobile device management (MDM) can specify a connection for each managed app and/or 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 does not.

iOS supports Always-on VPN, which can be configured for devices managed via MDM and supervised using Apple Configurator or the Device Enrollment Program. 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 now 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 128-bit AES 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.

iOS uses a randomized Media Access Control (MAC) address when conducting Preferred Network Offload (PNO) scans when a device is not associated with a Wi-Fi network *and* its processor is asleep. A device's processor goes to sleep shortly after the screen is turned off. PNO scans are run to determine if a user can connect to a preferred Wi-Fi network to conduct activity such as wirelessly syncing with iTunes.

iOS also uses a randomized MAC address when conducting enhanced Preferred Network Offload (ePNO) scans when a device is not associated with a Wi-Fi network *or* its processor is asleep. ePNO scans are run when a device uses Location Services for apps which 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 it's not connected to 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.

We've worked with Wi-Fi manufacturers to let them know that background scans use a randomized MAC address, and that neither Apple nor manufacturers can predict these randomized MAC addresses.

Wi-Fi MAC address randomization is not supported on iPhone 4s.

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 1.5)
- Phone Book Access Profile (PBAP)
- 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, see 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.

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. Certificated-based authentication is also supported. SSO support is based on the open source Heimdal project.

The following encryption types are supported:

- AES128-CTS-HMAC-SHA1-96
- AES256-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 servers 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/or Safari web URLs should be allowed to use SSO.

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 Yosemite 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 if they want to be able 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 sharing 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 the Everyone mode, the same process is used but if a match in Contacts is not found, the receiving devices are shown in the AirDrop sending sheet with a silhouette and 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 a mobile device management solution.

Apple Pay

With Apple Pay, users can use supported iOS devices and Apple Watch to pay in an easy, secure, and private way. 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, rewards, and store cards and to make payments with Apple Pay. Users can view their cards and additional information about their card issuer, 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.

Secure Enclave: On iPhone and iPad, the Secure Enclave manages the authentication process and enables a payment transaction to proceed. It stores fingerprint data for Touch ID.

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 directly without going through the application processor.

Apple Pay Servers: The Apple Pay Servers manage the state of credit and debit cards in Wallet and the Device Account Numbers stored in the Secure Element. They communicate both with the device and with the payment network 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 payment applets certified by the payment networks. Credit or debit card data is sent from the payment network or card issuer encrypted to these payment applets using keys that are known only to the payment network and the payment applets' security domain. This data is stored within these payment 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.

Once payment is authorized by the card holder using Touch 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 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 are routed to the application processor, but only after encryption by the Secure Element to the Apple Pay Server.

Credit and debit card provisioning

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

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 Apple Pay. These client-server sessions are encrypted using SSL.

Full card numbers are not 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 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 Pay 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. Provisioning a card for 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.

There are two ways to provision a credit or debit card into Apple Pay:

- Adding a credit or debit card manually to Apple Pay
- Adding credit or debit cards on file from an iTunes Store account to Apple Pay

Adding a credit or debit card manually to Apple Pay

To add a card manually, including store cards, the name, credit 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 iSight camera. When the camera captures the card information, Apple attempts to populate the name, card number, and expiration date. The photo is never saved to the device or stored in the photo library. Once 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 will be able to 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 merchant's server. These keys are used to encrypt the card information that's sent to the merchant, which prevents the information from being read by the iOS device. The provisioning flow is similar to that used for manually added cards, described above, except that 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 the user will need to enter 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

The Secure Element will only allow a payment to be made after it receives authorization from the Secure Enclave, confirming the user has authenticated with Touch ID or the device passcode. Touch ID is the default method if available but the passcode can be used at any time instead of Touch ID. A passcode is automatically offered after three unsuccessful attempts to match a fingerprint and after five unsuccessful attempts, the passcode is required. A passcode is also required when Touch ID is not configured or not enabled for Apple Pay.

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. Even 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 is 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 is persisted while Apple Pay is enabled, protected by the Secure Enclave's encryption and anti-rollback mechanism. It is securely delivered to the Secure Element via the pairing key. On receipt of a new AR value, the Secure Element marks any previously added cards as deleted.

Credit and debit 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 logs 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.
- Wrist detection is turned off.

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

All payment transactions originating from the payment applets include a transaction-specific dynamic security code along with a Device Account Number. This one-time code is computed using a 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 of these codes, including the following:

- A random number generated by the payment applet
- Another random number generated by the terminal—in the case of an NFC transaction

  or
- Another random number generated by the server—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.

Contactless payments with Apple Pay

If iPhone is on and detects an NFC field, it will present the user with the relevant credit or debit card, or the default card, which is managed in Settings. The user can also go to the Wallet app and choose a credit or debit card, or when the device is locked, double-click the Home button.

Next, the user must authenticate using Touch 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.

Once 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 Apple Pay within apps

Apple Pay can also be used to make payments within iOS apps. When users pay in apps using Apple Pay, Apple receives encrypted transaction information and re-encrypts it with a merchant-specific key before it's sent to the merchant. Apple Pay retains anonymous transaction information such as approximate purchase amount. This information can't be tied back 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 or the device passcode. Once 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 sent by the Secure Element, 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 via 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. The app developer is able to 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.

Rewards cards

As of iOS 9, Apple Pay supports the Value Added Service (VAS) protocol for transmitting merchant rewards cards to compatible NFC terminals. The VAS protocol can be implemented on merchant terminals and uses NFC to communicate with supported Apple devices. The VAS protocol works over a short distance and is used to provide complementary services, such as transmission of rewards card information, as part of an Apple Pay transaction.

The NFC terminal initiates receiving the card information by sending a request for a card. If the user has a card with the store's identifier, the user is asked to authorize its use. If the merchant supports encryption, the card 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 card data, which is sent to the terminal. If the merchant does not support encryption, the user is asked to re-present the device to the terminal before the rewards card information is sent.

Suspending, removing, and erasing cards

Users can suspend Apple Pay on iPhone and iPad 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 Settings, 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 using 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 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, Spotlight 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 user name and password that is used to sign in to Apple services such as iCloud, iMessage, FaceTime, the iTunes Store, the iBooks Store, 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 cannot 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 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 does not look familiar, users are instructed to change their Apple ID password immediately.

Apple also offers two-step verification for Apple ID, which provides a second layer of security for the user's account. 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 his or her Apple ID account information, before signing in to iCloud, iMessage, FaceTime, and Game Center, and before making an iTunes Store, iBooks Store, or App Store purchase from a new device. This can prevent anyone from accessing a user's account, even if they know the password. 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.

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

iMessage

Apple iMessage is a messaging service for iOS devices 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 does not log messages or attachments, and their contents are protected by end-to-end encryption so no one but the sender and receiver can access them. Apple cannot 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's directory service (IDS), where they are associated with the user's phone number or email address, along with the device's APNs address.

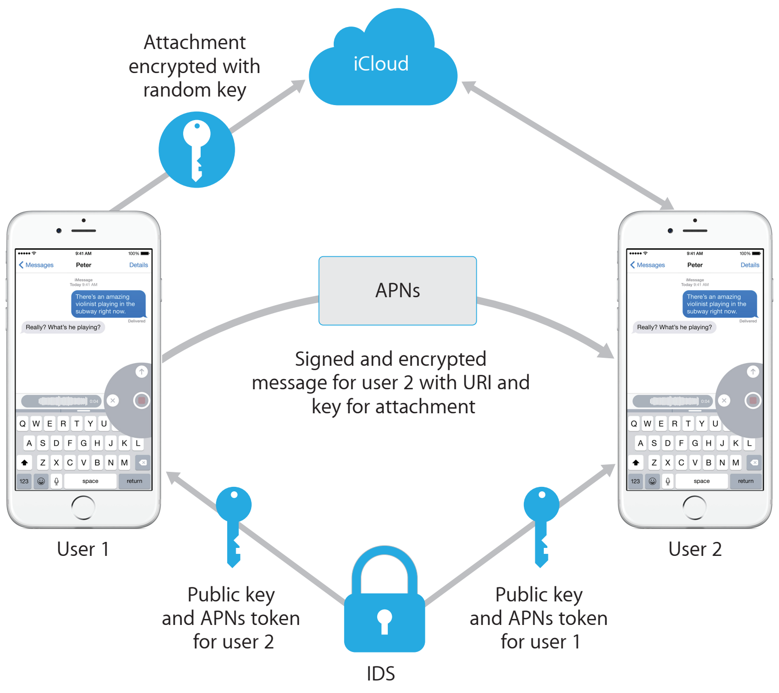
As users enable additional devices for use with iMessage, their encryption and signing public keys, APNs addresses, and associated phone numbers are added to the directory service. Users can also add more email addresses, which will be verified by sending a confirmation link. Phone numbers are verified by the carrier network and SIM. Further, all of the user's registered devices display an alert message when a new device, phone number, or email address is added.

How iMessage sends and receives messages

Users start a new iMessage conversation by entering an address or name. If they enter a phone number or email address, the device contacts the 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 the 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 128-bit key and encrypts the message with it using AES in CTR mode. 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, is not encrypted. Communication with APNs is encrypted using a forward-secret TLS channel.

APNs can only relay messages up to 4 KB or 16 KB 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 below.


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, if 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.

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 cannot decrypt the data.

FaceTime uses Internet Connectivity Establishment (ICE) to establish a peer-to-peer connection between devices. Using Session Initiation Protocol (SIP) messages, the devices verify their identity certificates and establish a shared secret for each session. The cryptographic nonces supplied by each device are combined to salt keys for each of the media channels, which are streamed via Secure Real Time Protocol (SRTP) using AES-256 encryption.

iCloud

iCloud stores a user's contacts, calendars, photos, documents, and more and keeps the information up to date across all of his or her 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 Backup, can be disabled by IT administrators via a configuration profile. 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, using third-party storage services, such as Amazon S3 and Windows Azure.

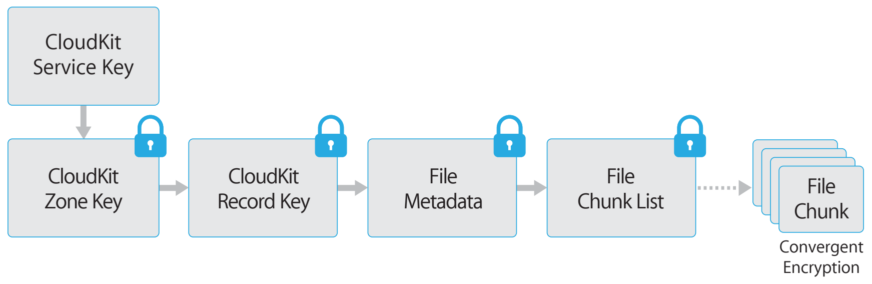
iCloud Drive

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 are not 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.



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:

- Information about purchased music, movies, TV shows, apps, and books, but not the purchased content itself
- Photos and videos in Camera Roll
- Contacts, calendar events, reminders, and notes
- Device settings
- App data
- PDFs and books added to iBooks but not purchased
- Call history
- Home screen and app organization
- iMessage, text (SMS), and MMS messages
- Ringtones
- HomeKit data
- HealthKit data
- Visual Voicemail

When files are created in Data Protection classes that are not 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.

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.

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 is not utilized for encryption so that 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.

Safari integration with iCloud Keychain

Safari can automatically generate cryptographically strong random strings for website passwords, which are stored in Keychain and synced to your 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 cannot read their contents.

iCloud Keychain

iCloud Keychain allows users to securely sync his or her 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.
- Third-party access to 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. A 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 P256) 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 cannot be read without knowing the user's iCloud password, and cannot 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, the new device notices in iCloud that the user has a previously established syncing circle that it is not 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 his or her 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 his or her 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. 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 is synced is encrypted specifically for the device it is being sent to. It cannot be decrypted by other devices or Apple. Additionally, the encrypted item is ephemeral in iCloud; it's overwritten with each new item that's synced.

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 is not 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 do not 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, the user is asked to create an iCloud Security Code. This code is required to recover an escrowed keychain. By default, the user is asked to provide a simple four-digit value for the security code. However, users can also 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 HSM (hardware security module) cluster that will store the escrow record. This becomes the user's iCloud Escrow Record.

If the user decided to accept a cryptographically random security code, instead of specifying his or her 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 is used to provide 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 clusters of hardware security modules (HSM). These clusters guard the escrow records. Each has a key that is used to encrypt the escrow records under their watch, as described previously.

To recover a keychain, users must authenticate with their iCloud account and password and respond to an SMS sent to their registered phone number. Once this is done, users must enter their iCloud Security Code. The HSM cluster verifies that a user knows his or her iCloud Security Code using Secure Remote Password protocol (SRP); the code itself is not sent to Apple. Each member of the cluster independently verifies that the user has not exceeded the maximum number of attempts that are allowed to retrieve his or her 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 owners of all keychains 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.

In order to facilitate Siri's 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, is sent to the server. This is so Siri can respond with the name or answer questions that only need an approximate location, such as those about the weather.

If a more precise location is necessary, for example, to determine 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 above. 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 are not 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. 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.

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 device 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 4.0 pairing out-of-band using the Apple Push Notification service (APNs). The individual messages are encrypted in a similar fashion to iMessage. Once the devices are paired, each will generate a symmetric 256-bit AES key that gets stored in the device's keychain. This key is used to 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 4.0 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 the Apple Push Notification service 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 webpages 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 used for shared web credentials. For details, refer to "Access to Safari saved passwords" in the Encryption and Data Protection section. The system must validate an app's domain name control before the app is permitted to accept user activity Handoff.

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

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

In the opposite direction, a native app may specify a fallback URL when a Handoff-receiving device does not 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 is complete.

iPhone Cellular Call Relay

When your Mac, iPad, or iPod is on the same Wi-Fi network as your iPhone, it can make and receive phone calls using your iPhone cellular connection. Configuration requires your 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 uses. Devices that are on the same network will present the incoming call notification UI. Upon answering the call, the audio will be seamlessly transmitted from your iPhone using a secure peer-to-peer connection between the two devices.

Outgoing calls will also be 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 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 SMS Message Forwarding is turned on, enrollment is verified on each device by entering a random six-digit numeric code generated by iPhone.

Once devices are linked, iPhone encrypts and forwards incoming SMS text messages to each device, utilizing the methods described in the iMessage section of this document. 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 and 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 signal 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) 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.

Spotlight Suggestions

Safari search and Spotlight search include search suggestions from the Internet, apps, iTunes, App Store, movie showtimes, locations nearby, and more.

To make suggestions more relevant to users, user context and search feedback with search query requests are sent to Apple. Context sent with search requests provides Apple with: i) the device's approximate location; ii) the device type (e.g., Mac, iPhone, iPad, or iPod); iii) the client app, which is either Spotlight or Safari; iv) the device's default language and region settings; v) the three most recently used apps on the device; and vi) an anonymous session ID. All communication with the server is encrypted via HTTPS.

To help protect user privacy, Spotlight Suggestions never sends exact location, instead blurring the location on the client before sending. The level of blurring is based on estimated population density at the device's location; for instance, more blurring is used in a rural location versus less blurring in a city center where users will typically be closer together. Further, users can disable the sending of all location information to Apple in Settings, by turning off Location Services for Spotlight Suggestions. If Location Services is disabled, then Apple may use the client's IP address to infer an approximate location.

The anonymous session ID allows Apple to analyze patterns between queries conducted in a 15-minute period. For instance, if users frequently search for "Café phone number" shortly after searching for "Café," Apple may learn to make the phone number more available in results. Unlike most search engines, however, Apple's search service does not use a persistent personal identifier across a user's search history to tie queries to a user or device; instead, Apple devices use a temporary anonymous session ID for at most a 15-minute period before discarding that ID.

Information on the three most recently used apps on the device is included as additional search context. To protect the privacy of users, only apps that are in an Apple-maintained whitelist of popular apps and have been accessed within the last three hours are included.

Search feedback sent to Apple provides Apple with: i) timings between user actions such as key-presses and result selections; ii) Spotlight Suggestions result selected, if any; and iii) type of local result selected (e.g., "Bookmark" or "Contact"). Just as with search context, the search feedback is not tied to any individual person or device.

Apple retains Spotlight Suggestions logs with queries, context, and feedback for up to 18 months. Reduced logs including only query, country, language, date (to the hour), and device-type are retained up to two years. IP addresses are not retained with query logs.

In some cases, Spotlight Suggestions may forward queries for common words and phrases to a qualified partner in order to receive and display the partner's search results. These queries are not stored by the qualified partner and partners do not receive search feedback. Partners also do not receive user IP addresses. Communication with the partner is encrypted via HTTPS. Apple will provide city-level location, device type, and client language as search context to the partner based on which locations, device types, and languages Apple sees repeated queries from.

Spotlight Suggestions can be turned off in Settings for Spotlight, for Safari, or for both. If turned off for Spotlight, then Spotlight is reverted to being a local on-device-only search client that does not transmit information to Apple. If turned off in Safari, the user's search queries, search context, and search feedback are not transmitted to Apple.

Spotlight also includes mechanisms for making local, on-device content searchable:

- The CoreSpotlight API, which allows Apple and third-party apps to pass indexable content to Spotlight.
- The NSUserActivity API, which allows Apple and third-party apps to pass information to Spotlight regarding app pages visited by the user.

Spotlight maintains an on-device index of the information it receives using these two methods, so that results from this data can be shown in response to a user's search, or automatically when Spotlight is launched. There is also an on-device federated search API, only available to Apple-provided apps, which allows Spotlight to pass user search queries to apps for processing, and receive their results.

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, 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 for enterprise use.

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

For details about each policy, see the Configuration Profile Key Reference documentation at 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 cannot be started until after the device has been unlocked by the user.

The pairing process requires the user to unlock the device and accept the pairing request from the host. 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 (see Escrow keybags in the Keybags section). 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 has not been used for more than six months, it expires. For more information, see 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.

A user can clear the list of trusted hosts by using the "Reset Network Settings" or "Reset Location & Privacy" options. For more information, see 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 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 passcodes meet 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

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 server can be removed—but doing so will also remove all managed configuration information, data, and apps.

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

Mobile device management (MDM)

iOS support for MDM allows businesses to securely configure and manage scaled iPhone and iPad 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 (APNs). For example, APNs is used to wake the device so it can communicate directly with its MDM server 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, and even remotely wipe or lock managed devices. For more information on mobile device management, see www.apple.com/iphone/business/it/management.html.

Device Enrollment Program

The Device Enrollment Program (DEP) provides a fast, streamlined way to deploy iOS devices that an organization has purchased directly from Apple or through participating Apple Authorized Resellers and carriers. The organization can automatically enroll devices in MDM without having to physically touch or prep the devices before users get them. The setup process for users can be further simplified by removing specific steps in the Setup Assistant, 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. For example, they can order the devices from Apple, configure all the management settings, and have the devices shipped directly to the user's home address. Once the device is unboxed and activated, the device enrolls in the organization's MDM—and all management settings, apps, and books are ready for the user.

The process is simple: After enrolling in the program, administrators log in to the program website, link the program to their MDM server, and "claim" the iOS devices purchased through Apple. The devices can then be assigned to users via MDM. Once a user has been assigned, any MDM-specified configurations, restrictions, or controls are automatically installed. For more information, see https://deploy.apple.com.

Note: The Device Enrollment Program is not available in all countries or regions.

Apple Configurator

In addition to MDM, Apple Configurator for OS X makes it easy for anyone to deploy iOS devices. Apple Configurator can be used to quickly configure large numbers of devices with apps, data, restrictions, and settings.

Supervision

During the setup of a device, an organization can configure a device to be supervised. Supervision denotes that a device is institutionally owned, which provides additional control over its configuration and restrictions. Devices can be supervised during setup through the Device Enrollment Program or Apple Configurator.

For more information on configuring and managing devices using MDM or Apple Configurator, see the iOS Deployment Reference at https://help.apple.com/deployment/ios.

For information about the additional controls for supervised devices, see the Configuration Profile Reference: https://developer.apple.com/library/ios/featuredarticles/iPhoneConfigurationProfileRef/iPhoneConfigurationProfileRef.pdf.

Device restrictions

Administrators can restrict device features by installing a configuration profile. Some of the restrictions available include:

- Allow app installs
- Allow trusting enterprise apps
- Allow use of camera
- Allow FaceTime
- Allow screenshots
- Allow voice dialing while locked
- Allow automatic sync while roaming
- Allow in-app purchases
- Allow syncing of recent Mail
- Force user to enter store password for all purchases
- Allow Siri while device is locked
- Allow use of iTunes Store
- Allow documents from managed sources in unmanaged destinations
- Allow documents from unmanaged sources in managed destinations
- Allow iCloud Keychain sync
- Allow updating certificate trust database over the air
- Allow showing notifications on Lock screen
- Force AirPlay connections to use pairing passwords
- Allow Spotlight to show user-generated content from the Internet
- Enable Spotlight Suggestions in Spotlight
- Allow Handoff
- Treat AirDrop as unmanaged destination
- Allow enterprise books to be backed up
- Allow notes and bookmarks in enterprise books to sync across the user's devices
- Allow use of Safari
- Enable Safari autofill
- Force Fraudulent Website Warning

- Enable JavaScript
- Limit ad tracking in Safari
- Block pop-ups
- Accept cookies
- Allow iCloud backup
- Allow iCloud document and key-value sync
- Allow iCloud Photo Sharing
- Allow diagnostics to be sent to Apple
- Allow user to accept untrusted TLS certificates
- Force encrypted backups
- Allow Touch ID
- Allow Control Center access from Lock screen
- Allow Today view from Lock screen
- Require Apple Watch wrist detection

Supervised-only restrictions

- Allow iMessage
- Allow removal of apps
- Allow manual install of configuration profiles
- Global network proxy for HTTP
- Allow pairing to computers for content sync
- Restrict AirPlay connections with whitelist and optional connection passcodes
- Allow AirDrop
- Allow Find My Friends modification
- Allow autonomous Single App Mode for certain managed apps
- Allow account modification
- Allow cellular data modification
- Allow host pairing (iTunes)
- Allow Activation Lock
- Prevent Erase All Content and Settings
- Prevent enabling restrictions
- Third-party content filter
- Single App mode
- Always-on VPN
- Allow passcode modification
- Allow Apple Watch pairing
- Allow automatic app downloads
- Allow keyboard prediction, autocorrection, spell check, and short cuts

For more information about restrictions, see https://developer.apple.com/library/ios/featuredarticles/iPhoneConfigurationProfileRef/iPhoneConfigurationProfileRef.pdf

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 via 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.

Find My iPhone and Activation Lock

If a device is lost or stolen, it's important to deactivate and erase the device. With iOS 7 or later, when Find My iPhone is turned on, the device can't be reactivated without entering the owner's Apple ID credentials. It's a good idea for an organization to either supervise its devices or have a policy in place for users to disable the feature so that Find My iPhone doesn't prevent the organization from assigning the device to another individual.

With iOS 7.1 or later, a compatible MDM solution can enable Activation Lock on supervised devices when a user turns on Find My iPhone. MDM administrators can manage Find My iPhone Activation Lock by supervising devices with Apple Configurator or the Device Enrollment Program. The MDM solution can then store a bypass code when Activation Lock is enabled, and later use this code to clear Activation Lock automatically when the device needs to be erased and assigned to a new user. See your MDM solution documentation for details.

Important: By default, supervised devices never have Activation Lock enabled, even if the user turns on Find My iPhone. However, an MDM server may retrieve a bypass code and permit Activation Lock on the device. If Find My iPhone is turned on when the MDM server 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.

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 diagnostic and usage services used by Apple to improve iOS, location-based Siri information, location-based context for Spotlight Suggestions searches, local traffic conditions, and frequently visited locations used to estimate travel times.

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 on iPhone 5s or later
- Social media accounts, such as Twitter and Facebook
- Microphone
- Camera
- HomeKit
- HealthKit
- Bluetooth sharing

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 MDM and those installed by the user.

Privacy policy

Apple's privacy policy is available online at https://www.apple.com/legal/privacy.

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. Safari offers safe browsing with support for Online Certificate Status Protocol (OCSP), EV certificates, and certificate verification warnings. Mail leverages certificates for authenticated and encrypted Mail by supporting S/MIME, which permits per-message S/MIME, so S/MIME users can choose to always sign and encrypt by default, or selectively control how individual messages are protected. iMessage and FaceTime also provide client-to-client encryption.

For third-party apps, the combination of required code signing, sandboxing, and entitlements gives users solid protection against viruses, malware, and other exploits that compromise the security of other platforms. The App Store submission process works to further shield users from these risks by reviewing every iOS app before it's made available for sale.

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 apple.com/support/security.

Glossary

Address space layout randomization (ASLR)	A technique employed by iOS to make the successful exploitation of 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 and later, the position of all system apps and libraries are randomized, along with all third-party apps compiled as position-independent executables.
Apple Push Notification service (APNs)	A worldwide service provided by Apple that delivers push notifications to iOS devices.
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."
ECID	A 64-bit identifier that's unique to the processor in each iOS device. 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's loaded by LLB, and in turn loads XNU, as part of the secure boot chain.
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.
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 (system, backup, escrow, or iCloud Backup) has the same format: • A header containing: – Version (set to 3 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)	Code that's invoked by the Boot ROM, and in turn loads iBoot, as part of the secure boot chain.
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.
Smart card	An integrated, embedded circuit that provides secure identification, authentication, and data storage.
System on a chip (SoC)	An integrated circuit (IC) that incorporates multiple components into a single chip. The Secure Enclave is an SoC within Apple's A7-or-later central processor.
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 cannot 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 cannot 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 is not 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 OS X operating systems. It's assumed to be trusted, and enforces security measures such as code signing, sandboxing, entitlement checking, and ASLR.