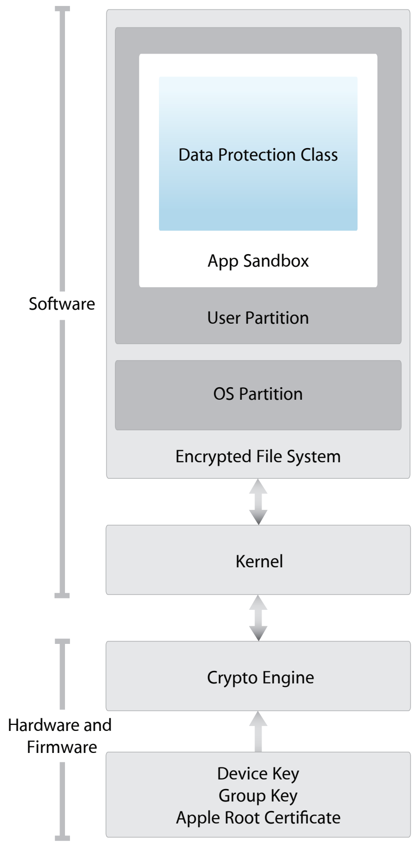
iOS Security

February 2014

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 OS, 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 OS security.

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 stringent security features, and they're 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.

- **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 protect 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.
- **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.

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 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 with the broadband subsystem.

For devices with an A7 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 even 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 http://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 it's 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 typically 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 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 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 chip. It utilizes its own secure boot and personalized software update separate from the application processor. It also 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. 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, tangled 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 tangled 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 purchase on behalf of the user. Communication between the A7 and the Touch ID sensor takes place over a serial peripheral interface bus. The A7 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 built into 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 built into iPhone 5s, making secure access to the device faster and easier. This forward-thinking technology reads fingerprints 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 rather 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 iPhone 5s so that it requires a passcode to unlock the device. When Touch ID scans and recognizes an enrolled fingerprint, iPhone 5s 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:

- iPhone 5s has just been turned on or restarted
- iPhone 5s has not been unlocked for more than 48 hours
- After five unsuccessful attempts to match a finger
- When setting up or enrolling new fingers with Touch ID
- iPhone 5s has received a remote lock command

When Touch ID is enabled, iPhone 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, iPhone 5s 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 users choose to authorize a purchase, authentication tokens are exchanged between the device and store. The token and 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 authentication and the data associated with the enrolled fingerprints are not available to other apps or third parties.

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 88-by-88-pixel, 500-ppi raster scan is temporarily stored in encrypted memory within the Secure Enclave while being vectorized for analysis, and then it's discarded after. 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 never leaves iPhone 5s, 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 iPhone 5s

On devices with an A7 processor, the Secure Enclave holds the cryptographic class keys for Data Protection. When a device locks, the keys for Data Protection class Complete are discarded, and files and keychain items in that class are inaccessible until the user unlocks the device by entering their passcode.

On iPhone 5s 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. 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 decrypted class keys are only held in memory, so they're lost if the device is rebooted. Additionally, as previously described, the Secure Enclave will discard the keys after 48 hours or 5 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. Along with the AES engine, SHA-1 is implemented in hardware, further reducing cryptographic operation overhead.

The device's unique ID (UID) and a device group ID (GID) are AES 256-bit keys fused into the application processor during manufacturing. No software or firmware can read them directly; they can see only the results of encryption or decryption operations performed using them. The UID is unique to each device and is not recorded by Apple or any of its suppliers. The GID is common to all processors in a class of devices (for example, all devices using the Apple A7 chip), and is used as an additional level of protection 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 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 gathered from interrupt timing during boot, and additionally from internal sensors once the device has booted.

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 sensitive data. Mail uses 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

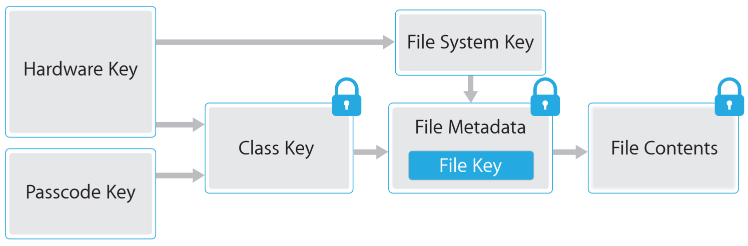
Every time a file on the data partition is created, Data Protection creates a new 256-bit key (the "per-file" key) and gives it to the hardware AES engine, which uses the key to encrypt the file as it is written to flash memory using AES CBC mode. The initialization vector (IV) is the output of a linear feedback shift register (LFSR) 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.

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

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 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.

Passcodes

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

The passcode is "tangled" 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 on iPhone 5s 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.

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.

To further discourage brute-force passcode attacks, the iOS interface enforces escalating time delays after the entry of an invalid passcode at the Lock screen. Users can choose to have the device automatically wiped if the passcode is entered incorrectly after 10 consecutive attempts. This setting is also available as an administrative policy through mobile device management (MDM) and Exchange ActiveSync, and can also be set to a lower threshold.

On a device with an A7 processor, the key operations are performed by the Secure Enclave, which also enforces a 5-second delay between repeated failed unlocking requests. This provides a governor against brute-force attacks in addition to safeguards enforced by iOS.

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 as follows.

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.

The Mail app implements Complete Protection for messages and attachments. App launch images and location data are also stored with Complete Protection.

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). Along with the usual per-file key, Data Protection generates a file public/private key pair. A shared secret is computed using the file's private key and the Protected Unless Open class public key, whose corresponding private key is protected with the user's passcode and the device UID. The per-file key is wrapped with the hash of this shared secret and stored in the file's metadata along with the file's public key; the corresponding private key is then wiped from memory. As soon as the file is closed, the per-file key is also 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-disk 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" and the "application-identifier" entitlement. 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. The prefix requirement is enforced through code signing and Provisioning Profiles.

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

Apps that utilize background refresh services in iOS 7 are required to use `kSecAttrAccessibleAfterFirstUnlock` for keychain items that need to be accessed during background updates.

Each keychain class has 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.

Components of a keychain item
Along with the access group, each keychain item contains administrative metadata (such as "created" and "last updated" time stamps). 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
- 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.

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 the OS. 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 certificates	Always, non-migratory
VPN passwords	After first unlock
LDAP, CalDAV, CardDAV	After first unlock
Social network account tokens	After first unlock
Home sharing password	When unlocked
Find My iPhone token	Always
iTunes backup	When unlocked, non-migratory
Voicemail	Always
Safari passwords	When unlocked
Bluetooth keys	Always, non-migratory
Apple Push Notification Service Token	Always, non-migratory
iCloud certificates and private key	Always, non-migratory
iCloud token	After first unlock
iMessage keys	Always, non-migratory
Certificates and private keys installed by Configuration Profile	Always, non-migratory
SIM PIN	Always, non-migratory

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 keychain 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 a passcode. The System keybag is the only keybag stored on the device. 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 are accessible, having 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 can be attempted on the Backup keybag. This threat can be mitigated with a sufficiently strong password.

If a user chooses to not 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 and passes it to the host. The Escrow keybag contains exactly the same class keys used on the device, protected by a newly generated key. This key is needed to unlock the Escrow keybag, and is stored on the device in the Protected Until First User Authentication class. This is why the device passcode must be entered before backing up with iTunes for the first time after a reboot.

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.

FIPS 140-2

The cryptographic modules in iOS 7 have been validated to comply with U.S. Federal Information Processing Standard (FIPS) 140-2 Level 1. This validates the integrity of cryptographic operations in Apple apps and third-party apps that properly utilize iOS cryptographic services. Bluetooth services have not been validated. For more information, see http://support.apple.com/kb/HT5808.

App Security

Apps are among the most critical elements of a modern OS 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, cannot execute malicious code, and are sandboxed to protect user data at all times. 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.

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 iOS Developer Enterprise Program (iDEP) with a D-U-N-S number. Apple approves applicants after verifying their identity and eligibility. Once an organization becomes a member of iDEP, 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. In-house apps also check to ensure the signature is valid at runtime. Apps with an expired or revoked certificate will not 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 application programming interfaces (APIs) and services 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 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. Apps can't share data directly with each other; sharing can be implemented only by both the receiving and sending apps using custom URL schemes, or through shared keychain access groups.

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.

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 ensure the highest level of protection in their apps. Data Protection is available for file and database APIs, including NSFileManager, CoreData, NSData, and SQLite.

As of iOS 7, third-party apps that do not opt-in to a specific data protection class receive Protected Until First User Authentication by default. For devices that were upgraded from an earlier release to iOS 7, apps that were already installed at the time of the upgrade continue to use No Protection unless they specifically adopt a specific Data Protection class.

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 accessory communicates with an iOS device using a Lightning connector cable, or via Wi-Fi or 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 Siri hands-free mode over Bluetooth. The 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 video streams utilize the MFi-SAP (Secure Association Protocol), which encrypts communication between the accessory and device using ECDH key exchange (Curve25519) with 2048-bit RSA keys and AES-128 in CTR mode.

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.

SSL, TLS

iOS supports Secure Socket Layer (SSL v3) as well as Transport Layer Security (TLS v1.0, TLS v1.1, TLS v1.2) and DTLS. Safari, Calendar, Mail, and other Internet applications 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.

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:

- Juniper Networks, Cisco, Aruba Networks, SonicWALL, Check Point, Palo Alto Networks, Open SSL, and F5 Networks SSL-VPN using the appropriate client app from the App Store. These apps provide user authentication for the built-in iOS support.
- Cisco IPSec with user authentication by Password, RSA SecurID or CRYPTOCard, and machine authentication by shared secret and certificates. Cisco IPSec supports VPN On Demand for domains that are specified during device configuration.
- 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.

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

iOS 7 introduces 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.

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.

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 http://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 session 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. 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 be whitelisted to also 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 (BTLE) and Apple-created peer-to-peer Wi-Fi technology to send files and information to nearby devices.

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 BTLE. 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. 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. 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.

The Wi-Fi radio is used to communicate directly between devices without using any Internet connection or Wi-Fi Access Point.

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, 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.

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, 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 for signing. For each key pair, the private keys 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.

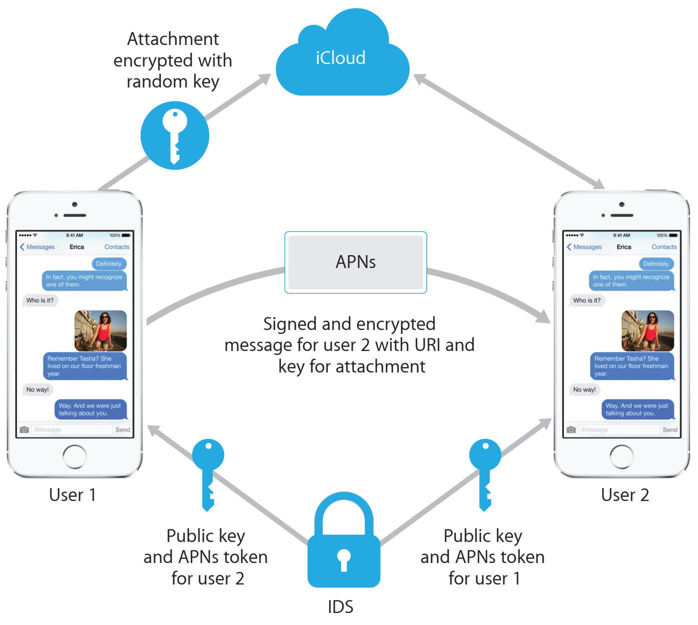
As users enable additional devices for use with iMessage, their public keys, APNs addresses, and associated phone numbers are added to the directory service. Users can also add more email addresses, which will be verified by sending a confirmation link. Phone numbers are verified by the carrier network 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 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 using AES-128 in CTR mode for each of the recipient's devices, signed using the sender's private key, and 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 TLS.

If the message text is too long, or if an attachment such as a photo is included, the attachment is encrypted using a random key and uploaded to iCloud. The key and URI (Uniform Resource Identifier) for the attachment are encrypted and signed, 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, iMessages are queued for delivery to offline devices. Messages are stored for up to seven 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 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.

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, perhaps to determine the location of nearby movie theaters for example, 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 in order to process the user's request. In any event, session information is discarded after 10 minutes of inactivity.

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.

iCloud

iCloud stores music, photos, apps, calendars, documents, and more, and automatically pushes them to all of a user's devices. 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. An iCloud account is configured via the Settings app by the user. iCloud features, including Photo Stream, Documents & Data, and Backup, can be disabled by IT administrators via a configuration profile.

The service is agnostic about what is being stored and handles all files the same way. There are two components for each file. The first is the file's metadata, which consists of its name, extension, and filesystem permission settings. The second component is the file's contents, which are treated by iCloud simply 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 Backup

iCloud also backs up information—including device settings, app data, and text and MMS messages—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
- Device settings
- App data
- Home screen and app organization
- iMessage, text (SMS), and MMS messages
- Ringtones
- 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 as described in iCloud, above.

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

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

- A user's iCloud account is compromised.
- iCloud is compromised by an external attacker or employee.
- 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 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 their iCloud password. The elliptical key generation parameters are retrieved from iCloud and generate a key that is used to sign the application ticket. Finally, the application ticket is placed in iCloud.

When the first device sees that an application ticket has arrived, it displays a notice for the user to acknowledge that a new device is asking to join the syncing circle. The user enters their iCloud password, and the application ticket is verified as signed by a matching private key. This establishes that the person who generated the request to join the circle entered the user's iCloud password at the time the request was made.

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

How keychain syncing works

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 members. 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.

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 with a random key, and places it in the user's iCloud key value storage area. The random key used to encrypt the keychain 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 their own or using a four-digit value, no escrow record is necessary. Instead, the iCloud Security Code is used to wrap the random key directly.

In addition to establishing a security code, users must register a phone number. This 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, the user must authenticate with their iCloud account and password and respond to an SMS sent to their registered phone number. Once this is done, the user must enter their iCloud Security Code. The HSM cluster verifies that the user knows their 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 their record, as discussed below. If a majority agree, the cluster unwraps the escrow record and sends it to the user's device.

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

These policies are coded in the HSM firmware. The administrative access cards that permit the firmware to be changed have been destroyed. Any attempt to alter the firmware or access the private key will cause the HSM cluster to delete the private key. Should this occur, the 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.

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

In addition to providing the cryptographic protection discussed earlier, passcodes prevent unauthorized access to the device's UI. The iOS interface enforces escalating time delays after the entry of an invalid passcode, dramatically reducing the effectiveness of brute-force attacks via the Lock screen. Users can choose to have the device automatically wiped if the passcode is entered incorrectly after 10 consecutive attempts. This setting is available as an administrative policy and can also be set to a lower threshold through MDM and Exchange ActiveSync.

By default, the user's passcode can be defined as a four-digit PIN. Users can specify a longer, alphanumeric passcode by turning on Settings > General > Passcode > Complex 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/.

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:

- Passcode policies
- Restrictions on device features (disabling the camera, for example)
- Wi-Fi settings
- VPN settings
- Email 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 email or 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. 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, visit www.apple.com/iphone/business/it/management.html.

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 the settings, apps, and data. Devices that are initially configured using Apple Configurator can be "supervised," enabling additional settings and restrictions to be installed. Once a device is supervised with Apple Configurator, all available settings and restrictions can be installed over the air via MDM as well. For more information on configuring and managing devices using both Apple Configurator and MDM, refer to *Deploying iPhone and iPad: Apple Configurator*.

Device Restrictions

Administrators can restrict device features by installing a configuration profile. The following restrictions are available:

- Allow app installs
- Allow use of camera
- Allow FaceTime
- Allow screen capture
- Allow voice dialing
- Allow automatic sync while roaming
- Allow in-app purchases
- Allow syncing of Mail recents
- Force user to enter store password for all purchases
- Allow multiplayer gaming
- Allow adding Game Center friends
- Allow Siri
- Allow Siri while device is locked
- Allow use of YouTube
- Allow Passbook notifications while device is locked
- Allow use of iTunes Store
- Allow use of Safari
- Enable Safari autofill
- Force Fraudulent Website Warning
- Enable JavaScript
- Block pop-ups
- Accept cookies
- Allow iCloud backup
- Allow iCloud document and key-value sync
- Allow Photo Streams
- Allow Shared Photo Streams
- Allow diagnostics to be sent to Apple
- Allow user to accept untrusted TLS certificates
- Force encrypted backups
- Restrict media by content rating
- Allow Touch ID
- Allow Control Center access from Lock screen
- Allow Today view from Lock screen

Supervised Only Restrictions

- Allow iMessage
- Allow Game Center
- Allow iBooks Store
- Allow erotica from iBooks Store
- Allow removal of apps
- Enable Siri Profanity Filter
- Allow manual install of configuration profiles
- Allow installation 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 account modification
- Allow Cellular Data modification
- Allow Find My Friends
- Allow Host Pairing (iTunes)
- Allow Activation Lock

Remote Wipe

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

When remote wiping is triggered by MDM or iCloud, the device sends an acknowledgment and performs the wipe. For remote wiping 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.

Conclusion

A Commitment to Security

From hardware to encryption to device access, each component of the iOS security platform provides organizations with the resources they need to build enterprise-grade security solutions. Together, these components give iOS its industry-leading security features without making the device difficult to use.

Apple uses a consistent, integrated security infrastructure throughout iOS and the iOS apps ecosystem. Hardware-based storage encryption provides instant 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 email by supporting S/MIME. 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.

Apple is committed to incorporating proven encryption methods and creating modern mobile-centric privacy and security technologies to ensure that iOS devices can be used with confidence in any personal or corporate environment.

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.

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.
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 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 as the pseudorandom function (PRF) with a UID-derived key.
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.