Aishwarya Mallela

Mobile Application development is a process of creating software applications. These applicationsare designed to run on Mobile devices, such as smart phones and tablets. It uses a network connectionto work with remote computing resources.

Technology stack is a collection of software tools, programming languages, frameworks, APIs, and components that are required to develop a fully functional software.

The term mobile development stack or mobile stack refers to a combination of programming languages, platforms, frameworks, tools, UX/UI software

The mobile stack can be categorized into these components:

- Frontend - technologies that are used to develop the interface that interacts with end-users.
- Back-end - Tools and software needed to create the underlying processing on the server.
- Development - A consolidated platform that provides the necessary libraries and interfaces to build the app.
- Supporting - Various tools and technologies that improve the security, flexibility, and performance of the app.

Programming languages are Objective- C : It's the original language used to build the iOS app, and Apple is still providing support for it. Nevertheless, building an app with Objective-C isn't easy, and developers are prone to make mistakes with it.

For Swift UI Swift is a relatively newer language introduced in 2014, and developers are favoring it over Objective-C. Thanks to its safer syntax, fewer mistakes are committed and apps are developed in much less time. It's also easier to find developers adept in Swift as it's easier to learn and master.

Development tools include Xcode and AppCode

XCode is the official development tool by Apple, and it's your go-to platform whether you're using Swift or Objective-C. It has all the features needed to build a native iOS mobile app, including a visual interface builder.

AppCode is Developed by a 3rd party, AppCode is an open-source alternative to XCode. While it enables iOS mobile app development, AppCode lacks the features of XCode and to an extent, is still dependent on the latter.

UI Kit and Swift UI are the UI Frameworks which are mostly used for developing an iOS applciation.

Rohith Chittipolu

Smartphone apps are the center of most peoples' technology usage. They deal with a lot of private and sensitive user data like your personal health information or banking information. Protecting this data as well as possible is heavily important.

Chances are that your app handles private data that you don't want to end up in the wrong hands. Therefore, you need to make sure to store this data safely and make data transportation as secure as possible.

If you are developing iOS apps lots of security features are already provided by the OS. All iOS devices with an A7 processor or later also have a coprocessor called the Secure Enclave. It powers iOS security features in a hardware-accelerated way.

Apple's App Sandbox
All apps running on iOS run in a sandbox to make sure the app can only access data which is stored in the app's unique home directory. If an app wants to access data outside of its home directory it needs to use services provided by iOS, like the ones available for accessing iCloud data or the photo album. Therefore, no other app can read or modify data from your app.

Apple's App Sandbox is powered by UNIX's user permissions and makes sure that apps get executed with a less privileged "mobile" user. Everything outside the app's home directory is mounted read-only. All system files and resources are protected. The available APIs don't allow apps to escalate privileges in order to modify other apps or iOS itself.

For performing specific privileged operations an app needs to declare special entitlements. These entitlements get signed together with the app and are not changeable. Examples of services that need special entitlements are HealthKit or audio input. Some entitlements are even restricted to be only used if Apple gives you access to them. This includes services like CarPlay. They are stronger protected because misusing them could have fatal consequences.

Next to entitlements giving you special rights, apps can make use of the iOS extensions system. The OS has many points to be used by app extensions. App extensions are single-purpose executables bundled with the app. They run in their own address space and get controlled by the OS.

Additionally, iOS has methods to prevent memory-related security bugs. Address space layout randomization (ASLR) randomizes the assigned memory regions for each app on every startup. This makes the exploitation of memory-corruption-bugs much less likely. Also, memory pages are marked as non-executable with ARM's Execute Never (XN) feature to stop malicious code from being executed.

Data Protection API

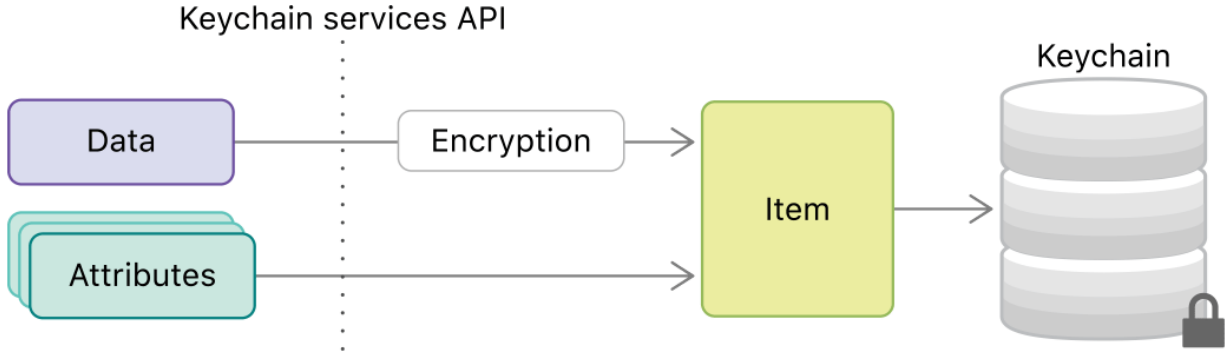
All iOS versions since iOS 4 have a built-in security feature called Data Protection. It allows an app to encrypt and decrypt the files stored in their app directory. The encryption and decryption processes are automatic and hardware-accelerated. Data Protection is available for file and database APIs, including NSFileManager, CoreData, NSData, and SQLite.

The feature is enabled by default but can be configured on a per-file basis to increase security. Every file can be configured to use one of 4 available protection levels. By default, all files are encrypted until the first user authentication but it might make sense to increase the protection level for certain data.

The four available protection levels include:

- **No protection**: The file is always accessible and not encrypted at all
- **Complete until first user authentication**: This is enabled by default and decrypts the file after the user unlocks their device for the first time. Afterward, the file stays decrypted until the device gets rebooted. Locking the device doesn't encrypt the data again.
- **Complete unless open**: The file is encrypted until the app opens the file for the first time. The decryption stays alive even when the device gets locked by the user.
- **Complete**: The file is only accessible when the device is unlocked.

Keychain



Putting data and attributes into a keychain (source)

The keychain is your secure place to store small chunks of data. It is a hardware-accelerated secure data storage that encrypts all of its contents. It is used by the system to store data like passwords and certificates but you as an app developer have also access to this data storage.

Your app or app group has its own space in the keychain and no other app has access to it. This way, you don't need to store encryption keys in your app and can rely on the system to provide the highest level of security.

The keychain is the secure key-value storage replacement for NSUserDefaults. NSUserDefaults are not encrypted at all and should be avoided for sensitive data.

For every keychain item, you can define specific access policies for accessibility and authentication. You can require user presence (requesting Face ID or Touch ID unlock) or that the biometric ID enrollment didn't change since the item was added to the keychain.

As an additional feature of the iOS Keychain, you can decide if you want to store the information in the local keychain which is only available on this specific device, or in the iCloud Keychain which gets synchronized across all Apple devices. This gives you the ability to share the information between your iPhone, iPad and Mac counterparts.

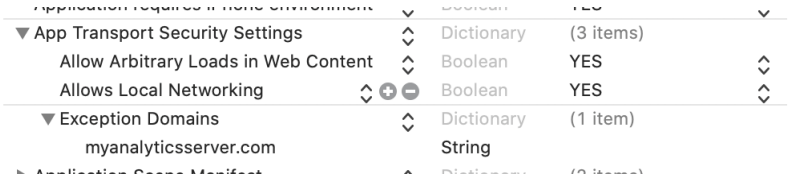
Even though the Keychain Services API should be used whenever possible, its interface is not exactly nice to use in Swift. If you plan to use such functionality more often in your codebase consider wrapping it with a nicer Swift API.

Best Practices for Secure Data Transportation
Next to storing the user data safely, you should make sure that the communication between your app and its remote counterparts is secured. This prevents attackers from collecting private data by sniffing the network traffic or by running malicious servers.

HTTPs
Most network communication is done over the HTTP protocol between a client and a server. By default, HTTP connections are not encrypted. It is easily possible for attackers to sniff data from your local network or to perform man-in-the-middle attacks.

Since iOS 9, there is a new feature called App Transport Security (ATS). It improves the security of network communication in your apps. ATS blocks insecure connections by default. It requires all HTTP connections to be performed using HTTPS secured with TLS.

ATS can be configured in many ways to loosen up these restrictions. You can, therefore, allow insecure HTTP connections for specific domains or change the minimum TLS version used for HTTPS.



If your app contains an in-app browser you should use the `NSAllowsArbitraryLoadsInWebContent` configuration which allows your users to browse the web normally and still makes sure that all other network connections in your app use the highest security standards.

SSL Pinning:
By default, HTTPS connections are verified by the system: it inspects the server certificate and checks if it's valid for this domain. This should make sure the connected server is not malicious. However, there are still ways for attackers to perform more complex man-in-middle attacks.

The system certificate trust validation checks if the certificate was signed by a root certificate of a trusted certificate authority. To circumvent this security mechanism attackers would need to explicitly trust another malicious certificate in the user's device settings or compromise a certificate authority. The attacker could then perform a man-in-the-middle attack to read all messages sent between client and server.

To prevent these kinds of attacks an app can perform additional trust validation of server certificates. This is called SSL or Certificate Pinning. You can implement SSL Pinning by including a list of valid certificates (or its public keys or its hashes) in your app bundle. The app can, therefore, check if the certificate used by the server is on this list and only then communicate with the server.

Implementing this validation from scratch should be avoided since implementation mistakes are very likely and lead to even more security vulnerabilities. We recommend using an Open Source framework like TrustKit. If you really need to implement this yourself read Apple's technical note on HTTPS Server Trust Evaluation.

Introducing SSL Pinning, unfortunately, introduces the risk of bricking your app. Since you hardcode the trusted certificates, the app itself needs to be updated if a server certificate expires. To avoid such a situation, pin the future certificates in the client app before releasing new server certificates.

Performing Key Agreement

In a lot of cases, you will need to perform some form of key exchange to exchange cryptographic keys over an insecure channel. With key agreement protocols like Elliptic-curve Diffie–Hellman (ECDH), you can derive a common secret.

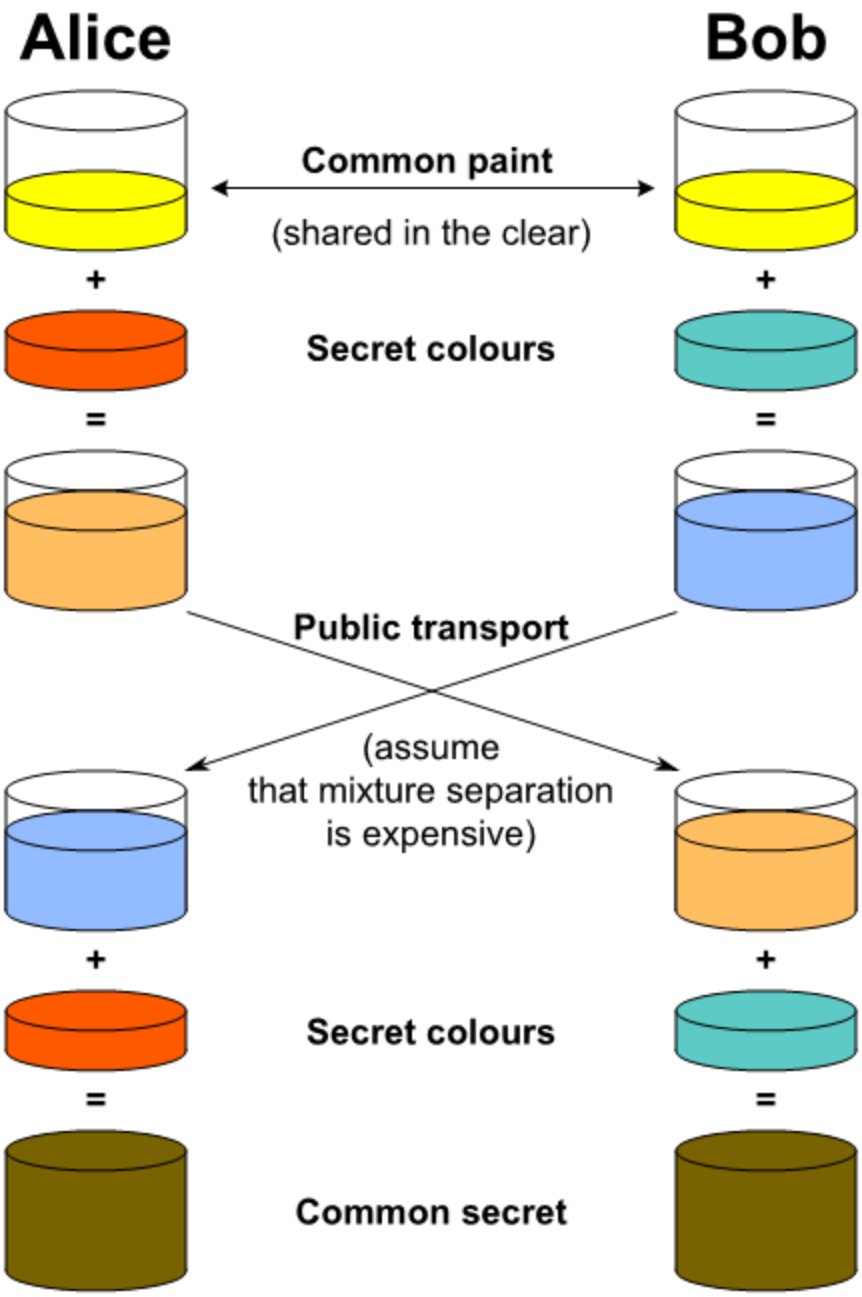


Illustration of the idea behind Diffie–Hellman key exchange (source)

Satwika Reddy Guntipally

Publishing an iOS App to App Store:
Apple is rather picky when it comes to accepting apps in the App Store. And indeed, Apple does make it much harder than Google when it comes to publishing apps. This does not mean it is impossible, however. All you will need is patience and, if possible, following the advice of somebody who has done it before.

You will need a Mac device or virtual machine to upload the app to App Store. The only way to upload apps is through XCode or Transporter and they both can only be installed in MacOS.

Apple Developer Portal

To upload an app-to-App Store you will first need to create an Apple Developer account in the Apple Developer Portal. Sign in with your Apple ID, fill in the enrollment form, pay the $100 registration fee and let us move on.

After your Developer Program Enrollment finishes you will be able to access developer resources from the Apple Developer Portal. Here you will need to create a few items:

- **Distribution certificate**: Certificate that identifies your team and allows you to submit apps to the App Store. Here is a great guide on how to create a distribution certificate manually (you will need to generate the certificate from a Mac).
- **App ID**: The identifier of your app. The Bundle ID must be the same as the bundle Identifier in your app binaries. When creating the App ID manually you will see a list of capabilities your app can add to your app (e.g. Push Notifications, Sign In with Apple, etc.). You can edit those capabilities later, so you do not need to configure them now.
- **Provision profile**: Only apps signed by Apple can be installed on an iOS device. This becomes an issue when developing, because you will want to test the app on a real device before submitting the app for review. Provisioning profiles are the solution to this problem; they are embedded in your app binaries and allow the app to run in certain devices before Apple signs it. A provisioning profile must be created for each app.

App Store Connect

After you have successfully created the items described above in the Apple Developer Portal, we can move on to App Store Connect (Former iTunes Connect), the platform where application binaries are uploaded.

Preparing for review:

Guidelines
Get in-depth details on the technical, content, and design criteria that we use for review, and learn about other key guidelines.

App Store Guidelines

Latest guidelines arranged into five clear sections: Safety, Performance, Business, Design, and Legal

1. Safety
When people install an app from the App Store, they want to feel confident that it is safe to do so—that the app does not contain upsetting or offensive content, will not damage their device, and is not likely to cause physical harm from its use. We have outlined the major pitfalls below, but if you are looking to shock and offend people, the App Store is not the right place for your app.

Objectionable Content

Apps should not include content that is offensive, insensitive, upsetting, intended to disgust, in exceptionally poor taste, or only plain creepy.

2. Performance

2.1 App Completeness

Submissions to App Review, including apps you make available for pre-order, should be final versions with all necessary metadata and fully functional URLs included; placeholder text, empty websites, and other temporary content should be scrubbed before submission. Make sure your app has been tested on-device for bugs and stability before you submit it and include demo account info (and turn on your back-end service!) if your app includes a login. If you offer in-app purchases in your app, make sure they are complete, up-to-date, and visible to the reviewer, or that you explain why not in your review notes. Please do not treat App Review as a software testing service. We will reject incomplete app bundles and binaries that crash or exhibit obvious technical problems.

2.2 Beta Testing

Demos, betas, and trial versions of your app do not belong on the App Store – use TestFlight instead. Any app submitted for beta distribution via TestFlight should be intended for public distribution and should comply with the App Review Guidelines. Note, however, that apps using TestFlight cannot be distributed to testers in exchange for compensation of any kind, including as a reward for crowd-sourced funding. Significant updates to your beta build should be submitted to TestFlight App Review before being distributed to your testers. To learn more, visit the TestFlight Beta Testing page.

2.3 Accurate Metadata

Customers should know what they are getting when they download or buy your app, so make sure all your app metadata, including privacy information, your app description, screenshots, and previews accurately reflect the app's core experience and remember to keep them up to date with new versions.

3. Business

There are many ways to monetize your app on the App Store. If your business model is not obvious, make sure to explain in its metadata and App Review notes. If we cannot understand how your app works or your in-app purchases are not immediately obvious, it will delay your review and may trigger a rejection. And while pricing is up to you, we will not distribute apps and in-app purchase items that are clear rip-offs. We will reject expensive apps that try to cheat users with irrationally unaffordable prices.

If we find that you have attempted to manipulate reviews, inflate your chart rankings with paid, incentivized, filtered, or fake feedback, or engage with third-party services to do so on your behalf, we will take steps to preserve the integrity of the App Store, which may include expelling you from the Apple Developer Program.

4. Design

Apple customers place a high value on products that are simple, refined, innovative, and easy to use, and that is what we want to see on the App Store. Producing a stately design is up to you, but the following are minimum standards for approval to the App Store. And remember that even after your app has been approved, you should update your app to ensure it remains functional and engaging to new and existing customers. Apps that stop working or offer a degraded experience may be removed from the App Store at any time.

5. Legal

Apps must comply with all legal requirements in any location where you make them available (if you're not sure, check with a lawyer). We know this stuff is complicated, but it is your responsibility to understand and make sure your app conforms with all local laws, not just the guidelines below. And of course, apps that solicit, promote, or encourage criminal or clearly reckless behavior will be rejected. In extreme cases, such as apps that are found to facilitate human trafficking and/or the exploitation of children, appropriate authorities will be notified.

Submitting for review

When you submit for review in App Store Connect, you can:

- Manage your submissions and communicate with App Review on the App Review page within My Apps. You can visit the App Review page at any time, even if you don't have active submissions or conversations.
- Submit items such as in-app events, custom product pages, and product page optimization tests without needing a new app version.
- Include multiple items in one submission.
- Remove items with issues from your submission and continue with items that were accepted by App Review.
- View a history of submissions created using the updated experience, including messages from App Review.

Review status

On average, 90% of submissions are reviewed in less than 24 hours. You'll be notified by email of status changes. You can also check the review status of your submission in the My Apps section of App Store Connect or on the App Store Connect app for iPhone and iPad. If your submission is incomplete, review times may be delayed, or your submission may not pass.

Avoiding common issues

We've highlighted some of the most common issues to help you better prepare before submitting for review

Crashes and bugs
Submit items for review only when they're complete and ready to be published. Make sure to thoroughly test on devices running the latest software and fix all bugs before submitting.

Broken links

All links in your app must be functional. A link to user support with up-to-date contact information and a link to your privacy policy is required for all apps.

Placeholder content
Finalize all images and text before submitting for review. Items that are still in progress and contain placeholder content are not ready to be distributed and cannot be approved.

Incomplete information

Enter all the details needed for review in the App Review Information section of App Store Connect. If some features require signing in, provide a valid demo account username and password. If there are special configurations to set, include the specifics. If features require an environment that is hard to replicate or require specific hardware, be prepared to provide a demo video or the hardware. Also, please make sure your contact information is complete and up to date

Aakanksha Reddy Sunkireddy

ADVANTAGES AND DISADVANTAGES OF i0S APP DEVELOPMENT OVER ANDROID:

Advantages of Native iOS App Development:

1)Apps are simple and fast to run as compared to other operating systems and application types, such including web or cross-platform applications.

2)Native iOS apps take full use of hardware and internal functions of the devices

3)Native applications get full support from App Store, as soon as they are accepted

4)Support from the App Store supplies complete security and security to iOS consumers

5)It's simple for customers to discover apps on App Store once they are approved and featured

6)Native app developers have more convenience due to availability of iOS unique software development kits (SDKs) and development tools

7)The iOS applications are closed sourced

8)App Store can handle your transactions on your behalf.

Disadvantages of Native iOS App Development:

1)Native iOS projects are substantially more expensive for developers, compared to cross-platform and web applications.

2)If the application supports a range of iOS devices, the costs of updating and maintenance are higher for the developer.

3)Being closed sourced, Mac OS X is needed to develop applications; they cannot be established on any other OS.

4)The process of getting an app authorized by the App Store is long and laborious; updates also need to be App Store authorized in order to be included there.

5)There is no guarantee that your app will be accepted and added to the App Store database

6)There is also no guarantee of fast popularity of application amongst iOS users

7)The worldwide market share for Apple has reduced to a small 11%.

8)Due to variety in iOS device type, version, and application version, it is a challenging process for developer to provide full support to all the application users.

9)If the application produces expenditures, 30 % should go to Apple's App Store, consisting of all the in-app purchases.