System Frameworks #WWDC16

Using StoreKit for In-App Purchases with Swift 3

Session 702

Dana DuBois App Store Engineering Manager

StoreKit

What's New

NEW

What's New

NEW

APIs in Swift

What's New

NEW

APIs in Swift

Subscriptions

Subscriptions

Subscriptions

Expanded categories

Subscriptions

Expanded categories

Increased proceeds

Subscriptions

Expanded categories

Increased proceeds

Territory pricing

Subscriptions

Expanded categories

Increased proceeds

Territory pricing

Preserve prices

Subscriptions

Expanded categories

Increased proceeds

Territory pricing

Preserve prices

Upgrades and downgrades

Subscriptions

Expanded categories

Increased proceeds

Territory pricing

Preserve prices

Upgrades and downgrades

Introducing Expanded Subscriptions in iTunes Connect — Pacific Heights — Tuesday 4:00PM

What's New

NEW

APIs in Swift

Subscriptions

What's New

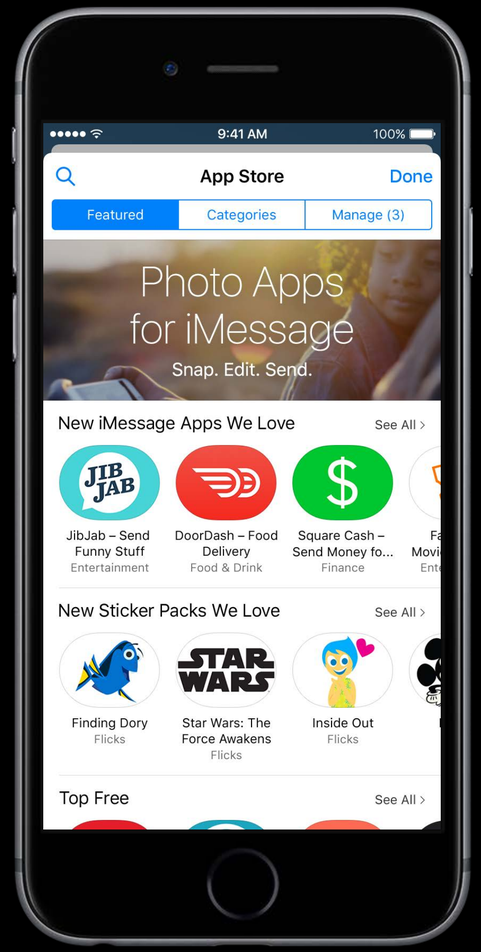
NEW

APIs in Swift

Subscriptions

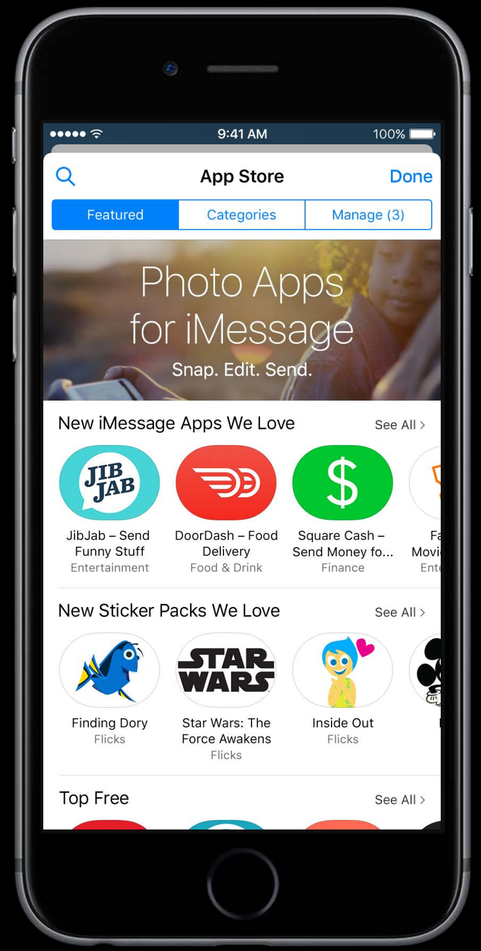
iMessage apps

iMessage Apps



iMessage Apps

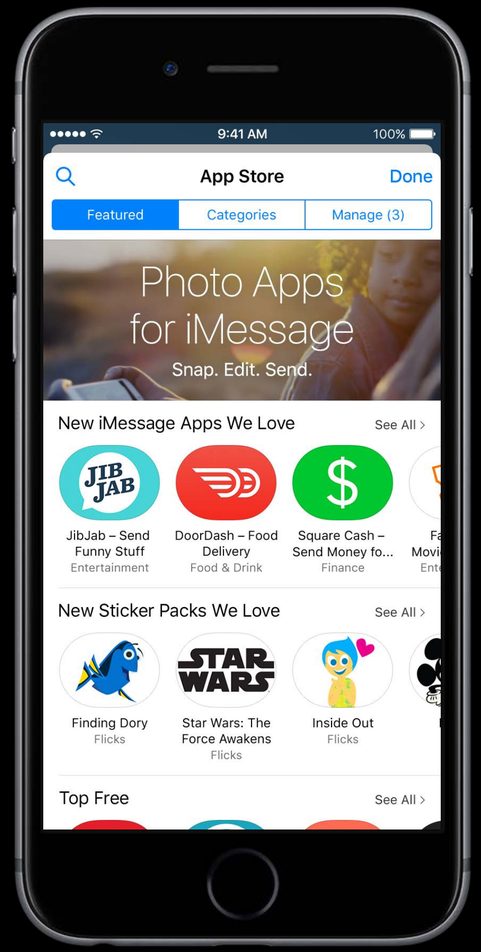
iMessage extensions will support In-App Purchases



iMessage Apps

iMessage extensions will support In-App Purchases

Same StoreKit APIs



In-App Purchase Overview

In-App Purchase Overview

Digital content or service bought in app

In-App Purchase Overview

Digital content or service bought in app

Not for physical goods

Types of In-App Purchases

Types of In-App Purchases

Consumable products

Types of In-App Purchases

Consumable products

Non-consumable products

Types of In-App Purchases

Consumable products

Non-consumable products

Non-renewing subscriptions

Types of In-App Purchases

Consumable products

Non-consumable products

Non-renewing subscriptions

Auto-renewing subscriptions

Implementing In-App Purchases

In-App Purchase Process

Load In-App Identifiers	Fetch Product Info	Show In-App UI	Make Purchase	Process Transaction	Make Asset Available	Finish Transaction

In-App Purchase Process

Load In-App Identifiers	Fetch Product Info	Show In-App UI	Make Purchase	Process Transaction	Make Asset Available	Finish Transaction

In-App Purchase Process

Load In-App Identifiers	Fetch Product Info	Show In-App UI	Make Purchase	Process Transaction	Make Asset Available	Finish Transaction

In-App Purchase Process

Load In-App Identifiers	Fetch Product Info	Show In-App UI	Make Purchase	Process Transaction	Make Asset Available	Finish Transaction

In-App Purchase Process

Load In-App Identifiers	Fetch Product Info	Show In-App UI	Make Purchase	Process Transaction	Make Asset Available	Finish Transaction

In-App Purchase Process

Load In-App Identifiers	Fetch Product Info	Show In-App UI	Make Purchase	Process Transaction	Make Asset Available	Finish Transaction

In-App Purchase Process

Load In-App Identifiers	Fetch Product Info	Show In-App UI	Make Purchase	Process Transaction	Make Asset Available	Finish Transaction

In-App Purchase Process

Load In-App Identifiers	Fetch Product Info	Show In-App UI	Make Purchase	Process Transaction	Make Asset Available	Finish Transaction

In-App Purchase Process

Load In-App Identifiers	Fetch Product Info	Show In-App UI	Make Purchase	Process Transaction	Make Asset Available	Finish Transaction

The Payment Queue

The Payment Queue

The center of your In-App Purchase implementation

- The only source of truth for state

The Payment Queue

The center of your In-App Purchase implementation

- The only source of truth for state

Rely on the queue, and only the queue

- For transactions in progress
- Payment status updates
- Download status

The Payment Queue

The center of your In-App Purchase implementation

- The only source of truth for state

Rely on the queue, and only the queue

- For transactions in progress
- Payment status updates
- Download status

Any and all transactions in the queue are valid and real

```swift
// Start Observing the Payment Queue

import UIKit
import StoreKit

@UIApplicationMain
class AppDelegate: UIResponder, UIApplicationDelegate, SKPaymentTransactionObserver {

    var window: UIWindow?

func application(application: UIApplication, didFinishLaunchingWithOptions
    launchOptions: [NSObject: AnyObject]?) -> Bool {
    SKPaymentQueue.defaultQueue().add(self);
    return true
}
```

```
// Start Observing the Payment Queue

import UIKit
import StoreKit

@UIApplicationMain
class AppDelegate: UIResponder, UIApplicationDelegate, SKPaymentTransactionObserver {

    var window: UIWindow?

    func application(application: UIApplication, didFinishLaunchingWithOptions
        launchOptions: [NSObject: AnyObject]?) -> Bool {
        SKPaymentQueue.defaultQueue().add(self);
        return true
    }
```

```
// Start Observing the Payment Queue

import UIKit
import StoreKit

@UIApplicationMain
class AppDelegate: UIResponder, UIApplicationDelegate, SKPaymentTransactionObserver {

    var window: UIWindow?

    func application(application: UIApplication, didFinishLaunchingWithOptions
        launchOptions: [NSObject: AnyObject]?) -> Bool {
        SKPaymentQueue.defaultQueue().add(self);
        return true
    }
```

```swift
// Start Observing the Payment Queue

import UIKit
import StoreKit

@UIApplicationMain
class AppDelegate: UIResponder, UIApplicationDelegate, SKPaymentTransactionObserver {

    var window: UIWindow?

func application(application: UIApplication, didFinishLaunchingWithOptions
    launchOptions: [NSObject: AnyObject]?) -> Bool {
    SKPaymentQueue.defaultQueue().add(self);
    return true
}
```

In-App Purchase Process

Load In-App Identifiers	Fetch Product Info	Show In-App UI	Make Purchase	Process Transaction	Make Asset Available	Finish Transaction

Load In-App Identifiers

Load In-App Identifiers

Options for storing the list of product identifiers

Load In-App Identifiers

Options for storing the list of product identifiers

- Baked-in product identifier

Load In-App Identifiers

Options for storing the list of product identifiers

- Baked-in product identifier
- Fetch from server

In-App Purchase Process

Load In-App Identifiers	Fetch Product Info	Show In-App UI	Make Purchase	Process Transaction	Make Asset Available	Finish Transaction

```
// Fetch Product Info

let request = SKProductsRequest(productIdentifiers: identifierSet)
request.delegate = self
request.start()
```

```
// Fetch Product Info

let request = SKProductsRequest(productIdentifiers: identifierSet)
request.delegate = self
request.start()
```

```
// Fetch Product Info

let request = SKProductsRequest(productIdentifiers: identifierSet)
request.delegate = self
request.start()
```

```
// Fetch Product Info

let request = SKProductsRequest(productIdentifiers: identifierSet)
request.delegate = self
request.start()
```

```
// Fetch Product Info

let request = SKProductsRequest(productIdentifiers: identifierSet)
request.delegate = self
request.start()
```

```swift
// Fetch Product Info

func productsRequest(_ request: SKProductsRequest, didReceive response: SKProductsResponse)
{
    for product in response.products {
        // Localized title and description
        product.localizedTitle
        product.localizedDescription
        // Price and locale
        product.price
        product.priceLocale
        // Content size and version (hosted)
        product.downloadContentLengths
        product.downloadContentVersion
    }
}
```

```swift
// Fetch Product Info

func productsRequest(_ request: SKProductsRequest, didReceive response: SKProductsResponse)
{
    for product in response.products {
        // Localized title and description
        product.localizedTitle
        product.localizedDescription
        // Price and locale
        product.price
        product.priceLocale
        // Content size and version (hosted)
        product.downloadContentLengths
        product.downloadContentVersion
    }
}
```

```swift
// Fetch Product Info

func productsRequest(_ request: SKProductsRequest, didReceive response: SKProductsResponse)
{
    for product in response.products {
        // Localized title and description
        product.localizedTitle
        product.localizedDescription
        // Price and locale
        product.price
        product.priceLocale
        // Content size and version (hosted)
        product.downloadContentLengths
        product.downloadContentVersion
    }
}
```

```swift
// Fetch Product Info

func productsRequest(_ request: SKProductsRequest, didReceive response: SKProductsResponse)
{
    for product in response.products {
        // Localized title and description
        product.localizedTitle
        product.localizedDescription
        // Price and locale
        product.price
        product.priceLocale
        // Content size and version (hosted)
        product.downloadContentLengths
        product.downloadContentVersion
    }
}
```

```swift
// Fetch Product Info

func productsRequest(_ request: SKProductsRequest, didReceive response: SKProductsResponse)
{
    for product in response.products {
        // Localized title and description
        product.localizedTitle
        product.localizedDescription
        // Price and locale
        product.price
        product.priceLocale
        // Content size and version (hosted)
        product.downloadContentLengths
        product.downloadContentVersion
    }
}
```

```swift
// Handle Events

func paymentQueue(_ queue: SKPaymentQueue, updatedTransactions transactions:
    [SKPaymentTransaction]) {
    for transaction in transactions {
        switch transaction.transactionState {
        case .purchased:
            // Validate the purchase
        }
    }
}
```

```
    // Handle Events

    func paymentQueue(_ queue: SKPaymentQueue, updatedTransactions transactions:
        [SKPaymentTransaction]) {
        for transaction in transactions {
            switch transaction.transactionState {
            case .purchased:
                // Validate the purchase
            }
        }
    }
```

```
    // Handle Events

    func paymentQueue(_ queue: SKPaymentQueue, updatedTransactions transactions:
        [SKPaymentTransaction]) {
        for transaction in transactions {
            switch transaction.transactionState {
            case .purchased:
                // Validate the purchase
            }
        }
    }
```

```
    // Handle Events

    func paymentQueue(_ queue: SKPaymentQueue, updatedTransactions transactions:
        [SKPaymentTransaction]) {
        for transaction in transactions {
            switch transaction.transactionState {
            case .purchased:
                // Validate the purchase
            }
        }
    }
```

```swift
// Handle Events

func paymentQueue(_ queue: SKPaymentQueue, updatedTransactions transactions:
    [SKPaymentTransaction]) {
    for transaction in transactions {
        switch transaction.transactionState {
        case .purchased:
            // Validate the purchase

        case .deferred:
            // Allow the user to continue to use the app
            // It may be some time before the transaction is updated
            // Do not get stuck in a modal “Purchasing...” state!
        }
    }
}
```

```swift
// Handle Events

func paymentQueue(_ queue: SKPaymentQueue, updatedTransactions transactions:
    [SKPaymentTransaction]) {
    for transaction in transactions {
        switch transaction.transactionState {
        case .purchased:
            // Validate the purchase

        case .deferred:
            // Allow the user to continue to use the app
            // It may be some time before the transaction is updated
            // Do not get stuck in a modal “Purchasing...” state!
        }
    }
}
```

Testing Deferred Transactions

Testing Deferred Transactions

Create a mutable payment

Testing Deferred Transactions

Create a mutable payment

Set the simulatesAskToBuyInSandbox flag

Testing Deferred Transactions

Create a mutable payment

Set the simulatesAskToBuyInSandbox flag

```
let payment = SKMutablePayment(product: product)
payment.simulatesAskToBuyInSandbox = true
SKPaymentQueue.defaultQueue().add(payment)
```

Handling Errors

Handling Errors

Not all errors are equal

Handling Errors

Not all errors are equal

Check the error code

- Don’t show an error alert unless necessary
- User canceling a payment will result in an error

Handling Errors

Not all errors are equal

Check the error code

- Don’t show an error alert unless necessary
- User canceling a payment will result in an error

Let StoreKit handle the transaction flow as much as possible

- Including asking for confirmation for purchase

Validate the Purchase

Working with receipts

Receipt Validation

Receipt Validation

On-device validation

- Unlock features and content within the app

Receipt Validation

On-device validation

- Unlock features and content within the app

Server-to-server validation

- Restrict access to downloadable content

Receipt Validation

On-device validation

- Unlock features and content within the app

Server-to-server validation

- Restrict access to downloadable content

Do not use online validation directly from the device!

The Receipt

The Receipt

Trusted record of App and In-App Purchases

The Receipt

Trusted record of App and In-App Purchases

Stored on device

The Receipt

Trusted record of App and In-App Purchases

Stored on device

Issued by the App Store

The Receipt

Trusted record of App and In-App Purchases

Stored on device

Issued by the App Store

Signed and verifiable

The Receipt

Trusted record of App and In-App Purchases

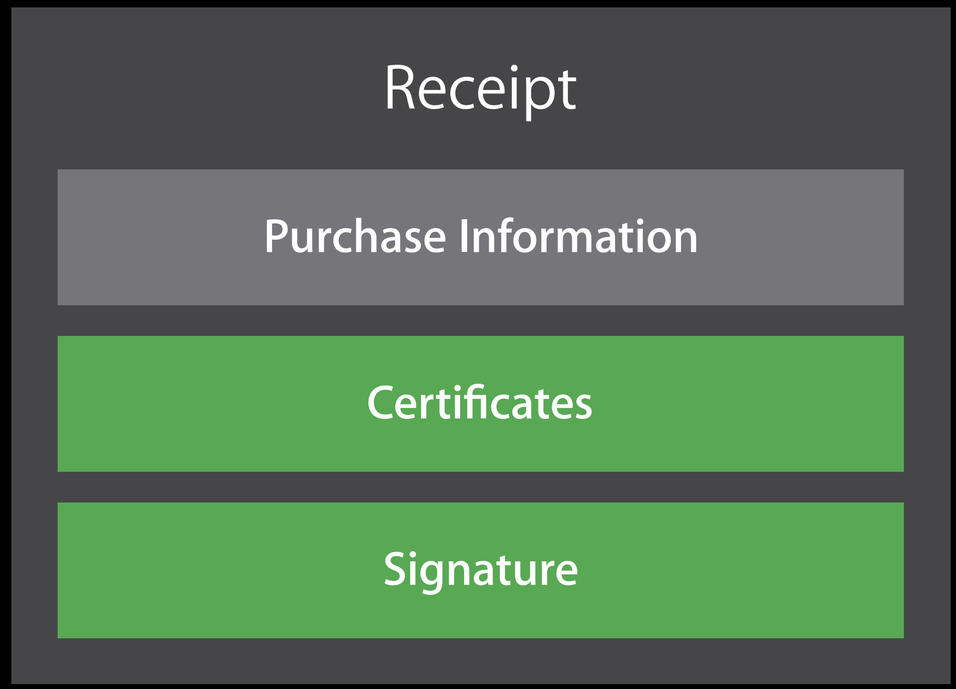
Stored on device

Issued by the App Store

Signed and verifiable

For your app, on that device only

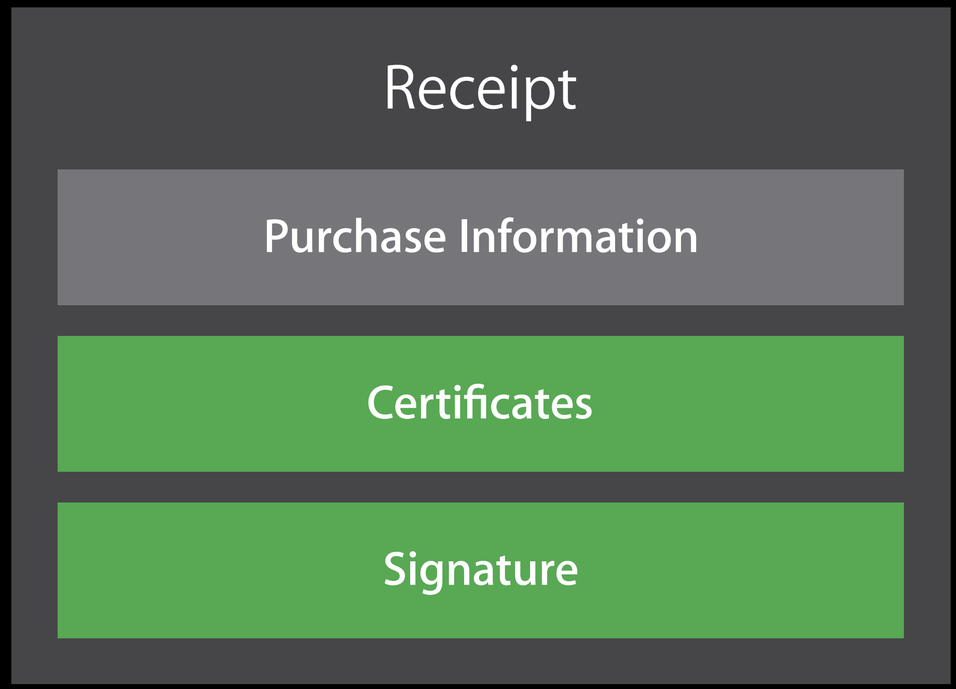
The Basics



The Basics

Stored in the App Bundle

- API to get the path



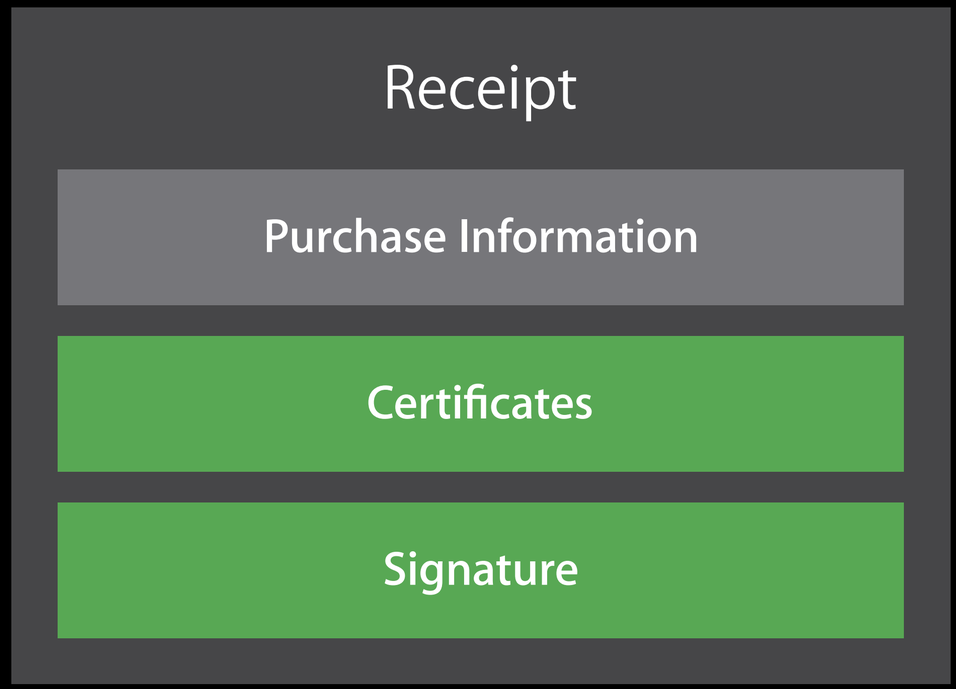
The Basics

Stored in the App Bundle

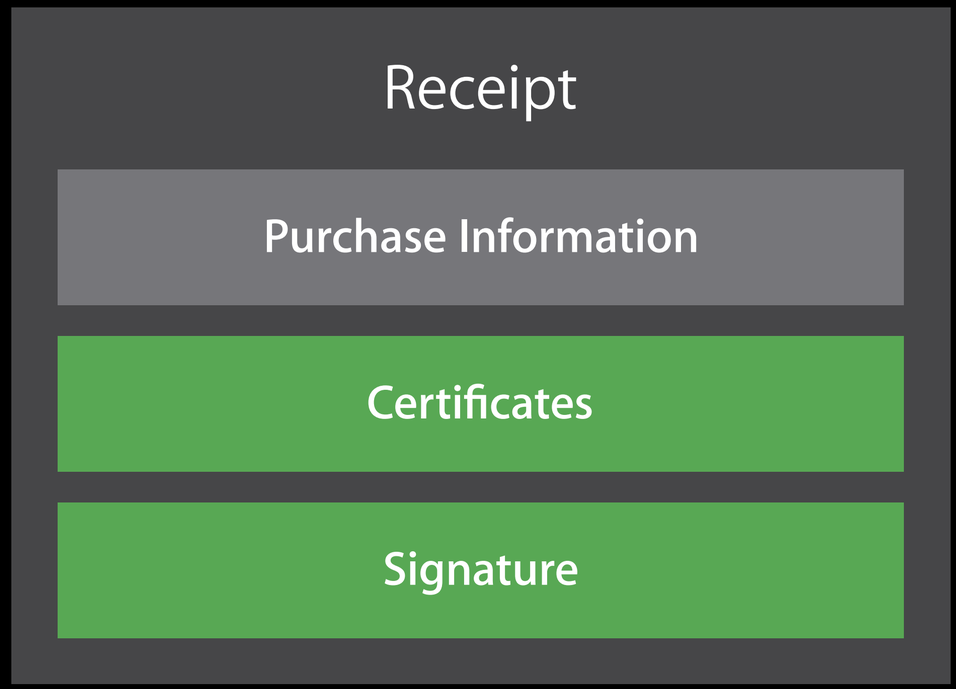
- API to get the path

Single file

- Purchase data
- Signature to check authenticity



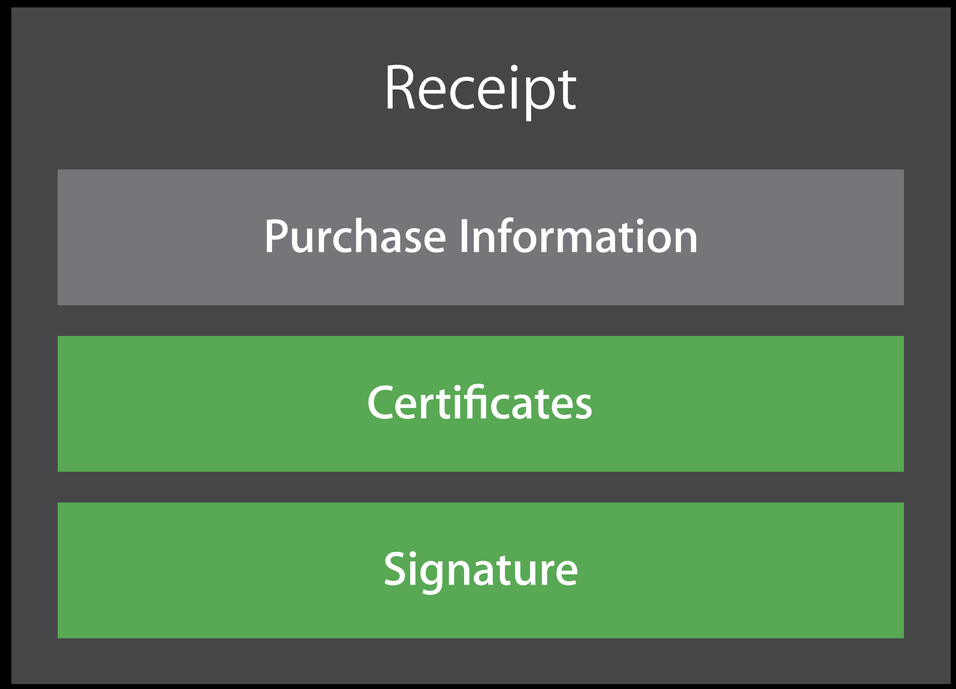
Standards



Standards

Signing

- PKCS#7 Cryptographic Container



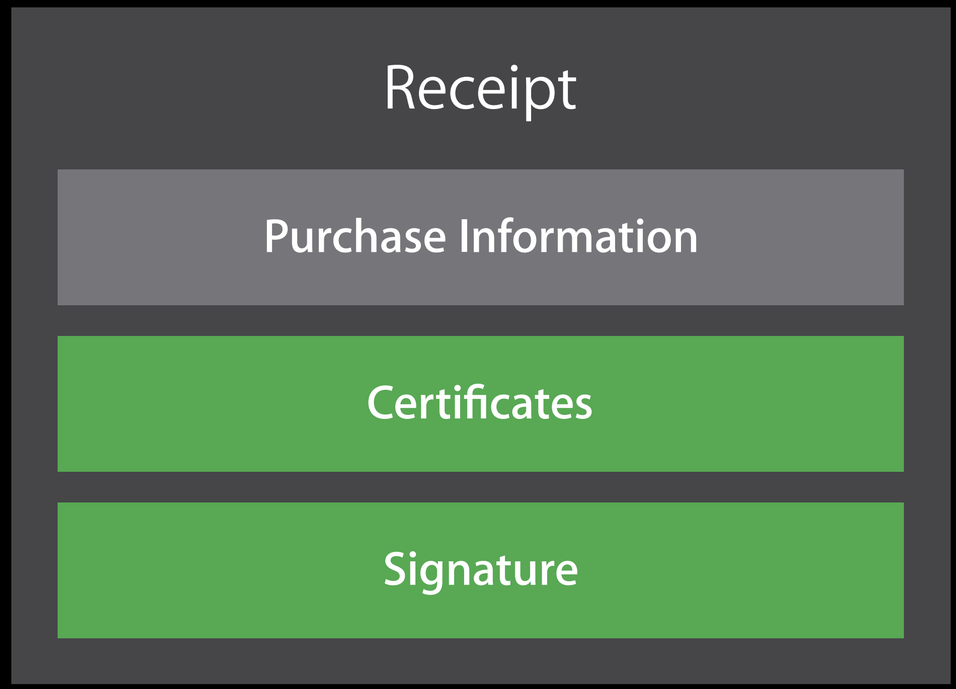
Standards

Signing

- PKCS#7 Cryptographic Container

Data Encoding

- ASN.1



Standards

Signing

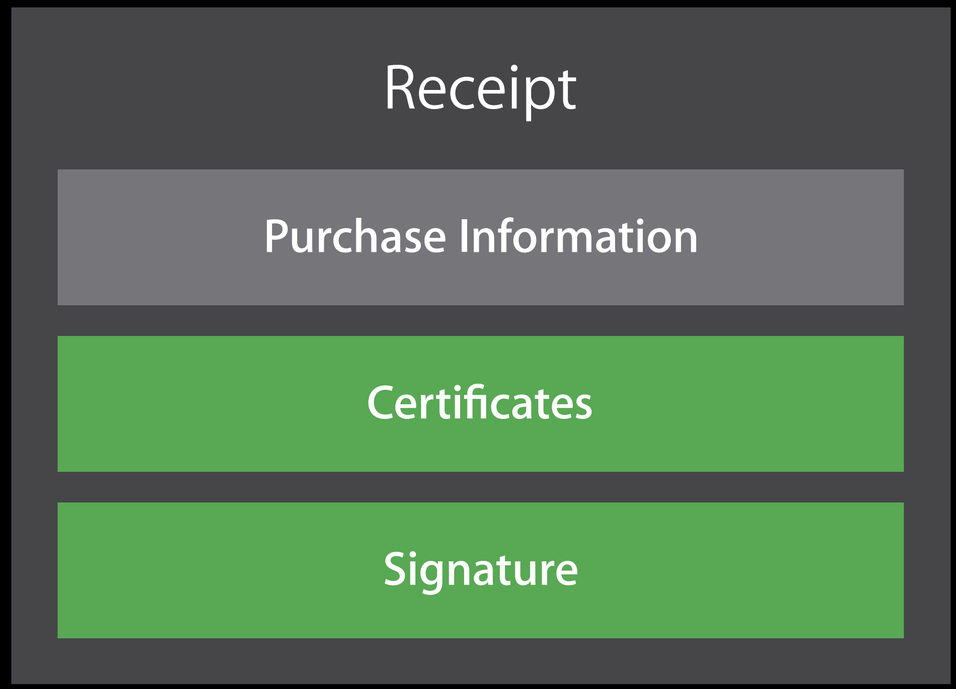
- PKCS#7 Cryptographic Container

Data Encoding

- ASN.1

Options for verifying and reading

- OpenSSL, asn1c, etc.
- Create your own



Getting Started

Getting Started

Locate the receipt using NSBundle API

Getting Started

Locate the receipt using NSBundle API

```
// Locate the file
let url = NSBundle.main().appStoreReceiptURL!

// Read the contents
let receipt = NSData(contentsOf: url)
```

Verification

Verification

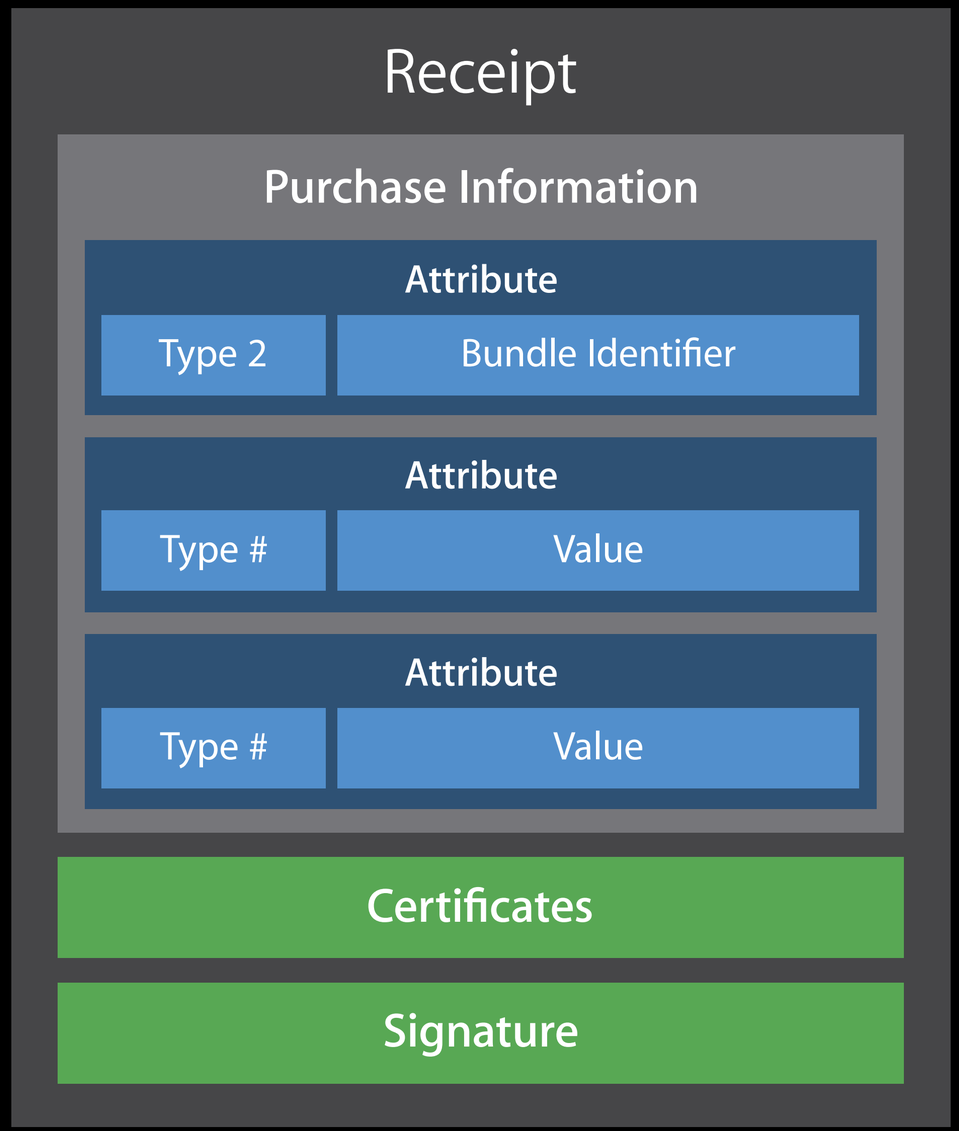
Do not check the expiry date on the certificate

Verification

Do not check the expiry date on the certificate

Do evaluate trust up to Root CA

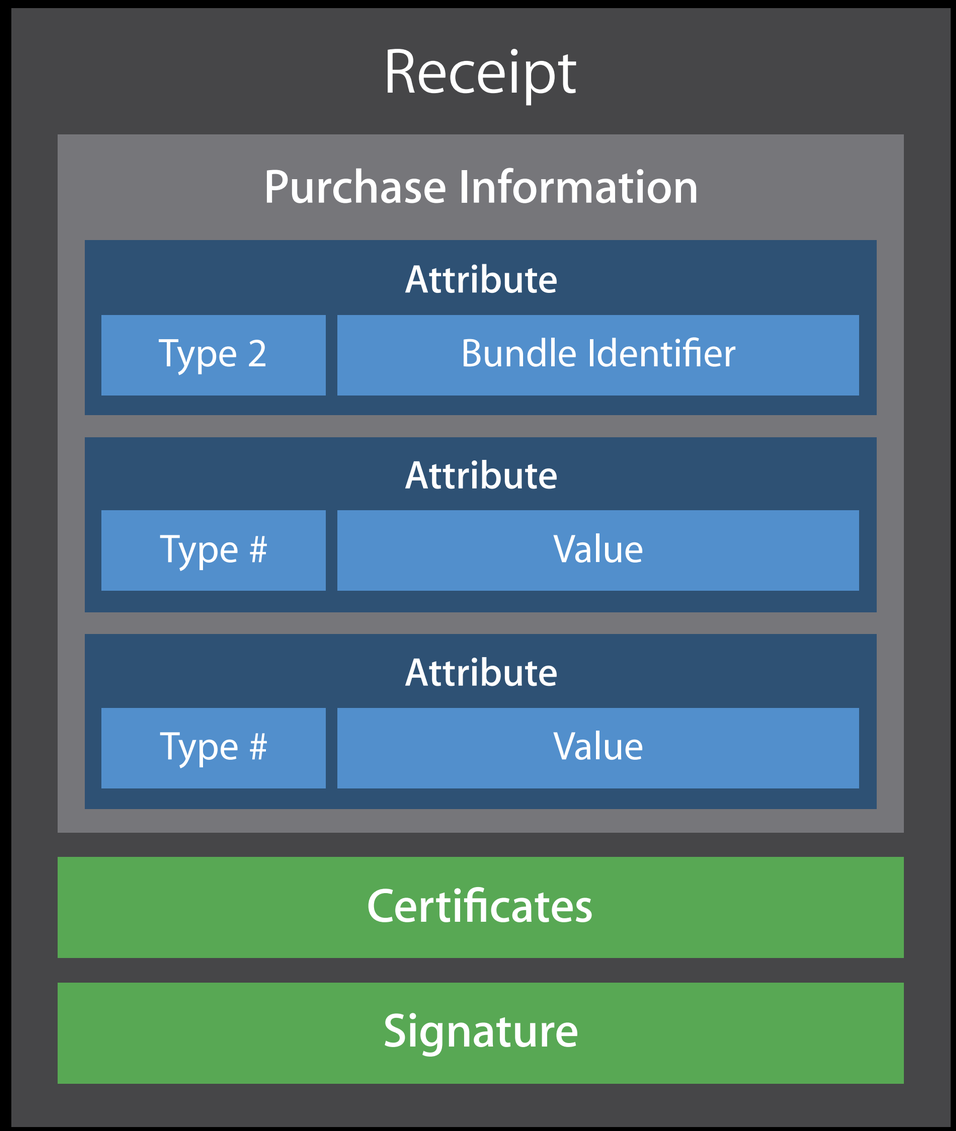
Receipt Payload



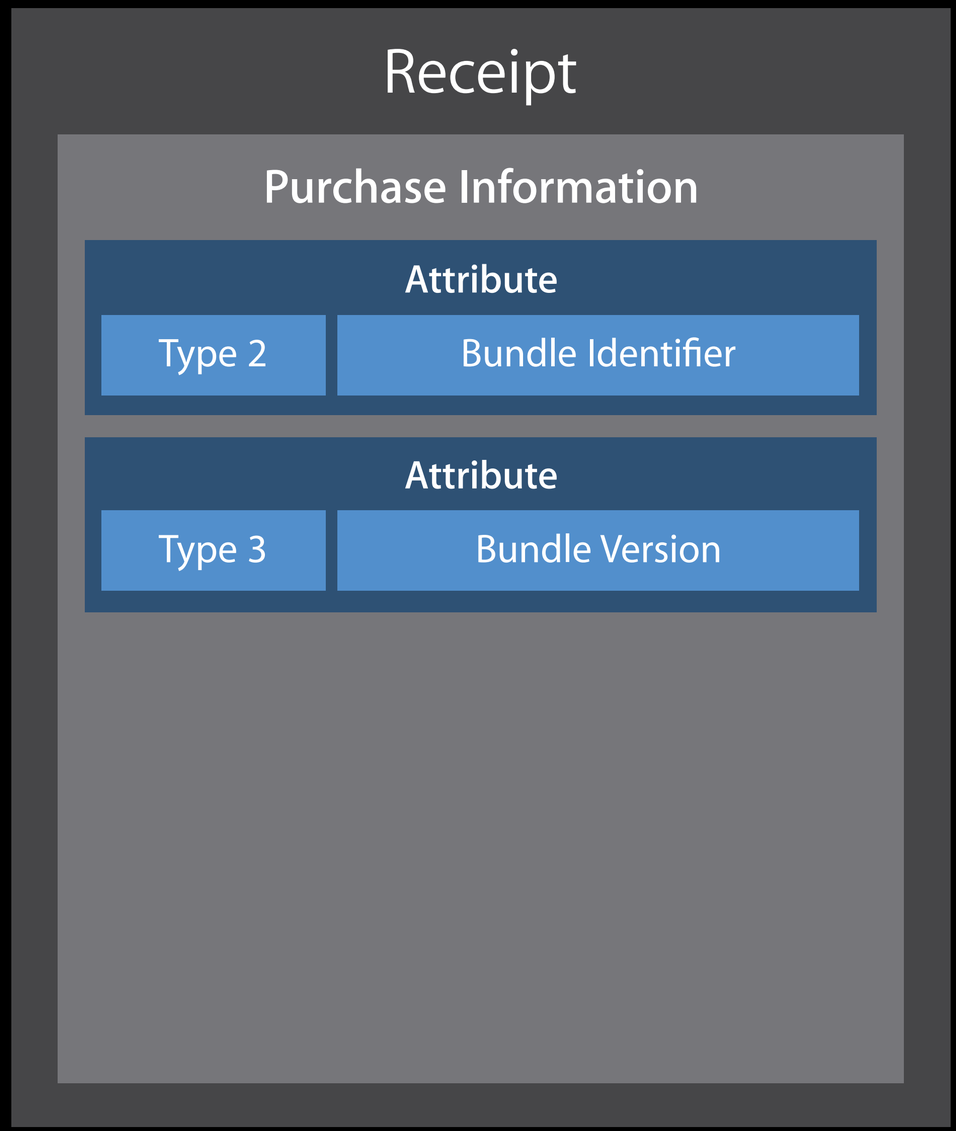
Receipt Payload

Series of attributes

- Type
- Value
- (Version)

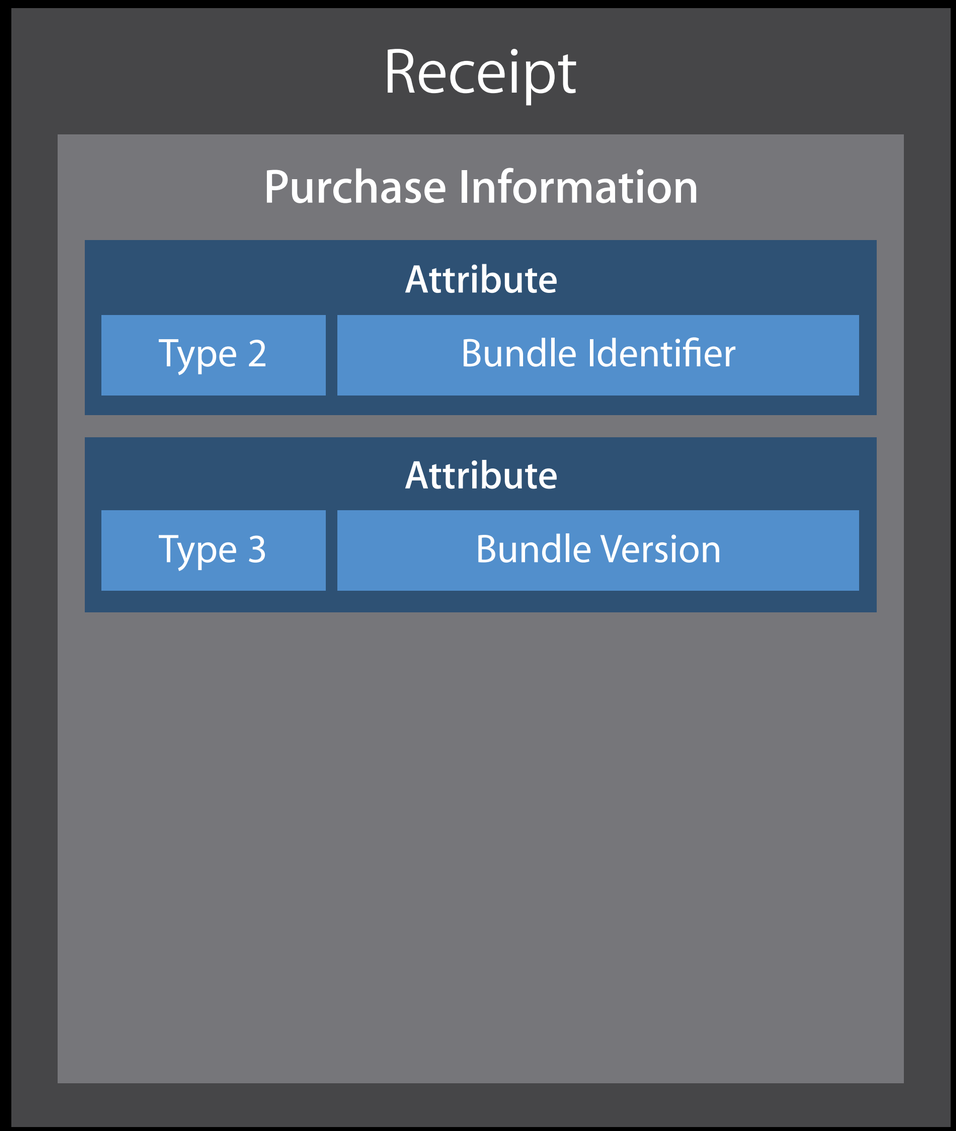


Verify Application



Verify Application

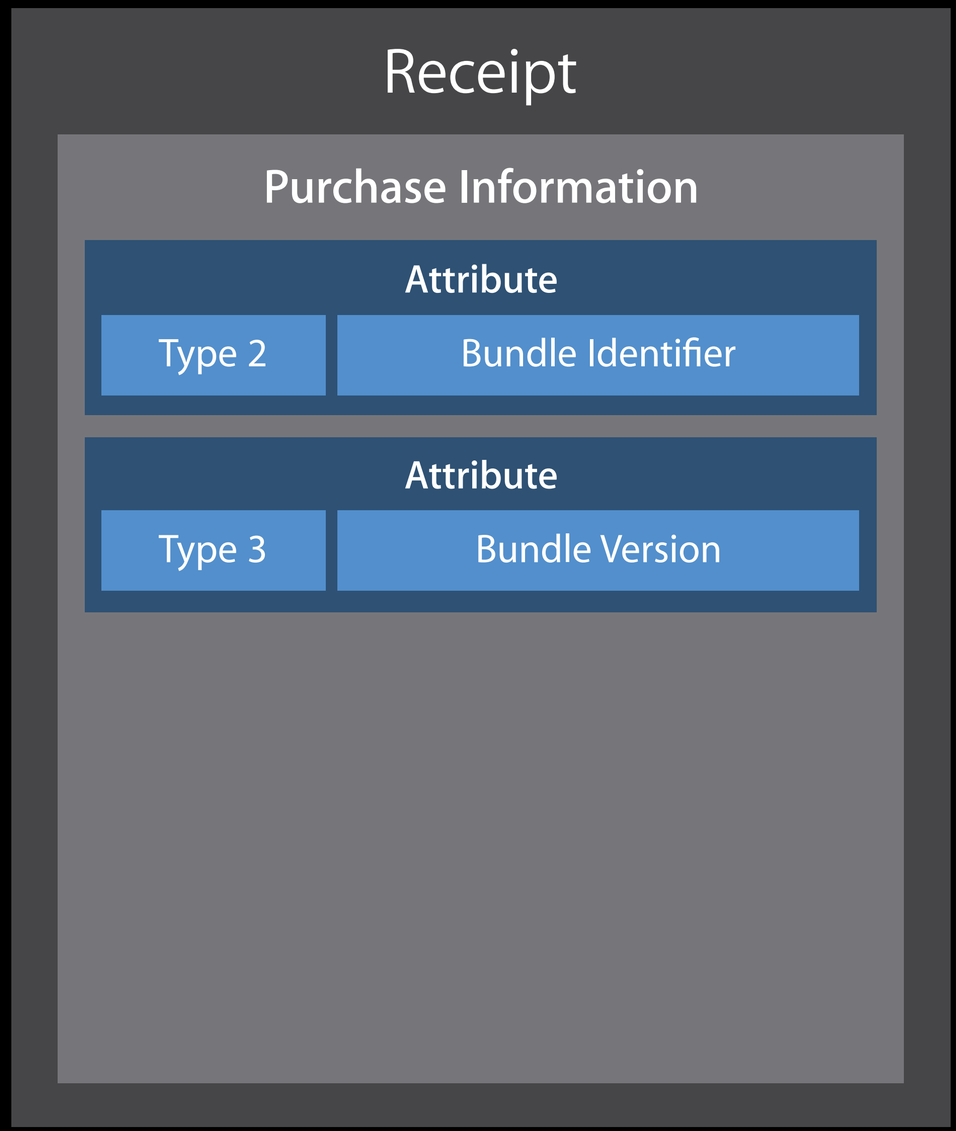
Check the Bundle Identifier



Verify Application

Check the Bundle Identifier

Check the Bundle Version



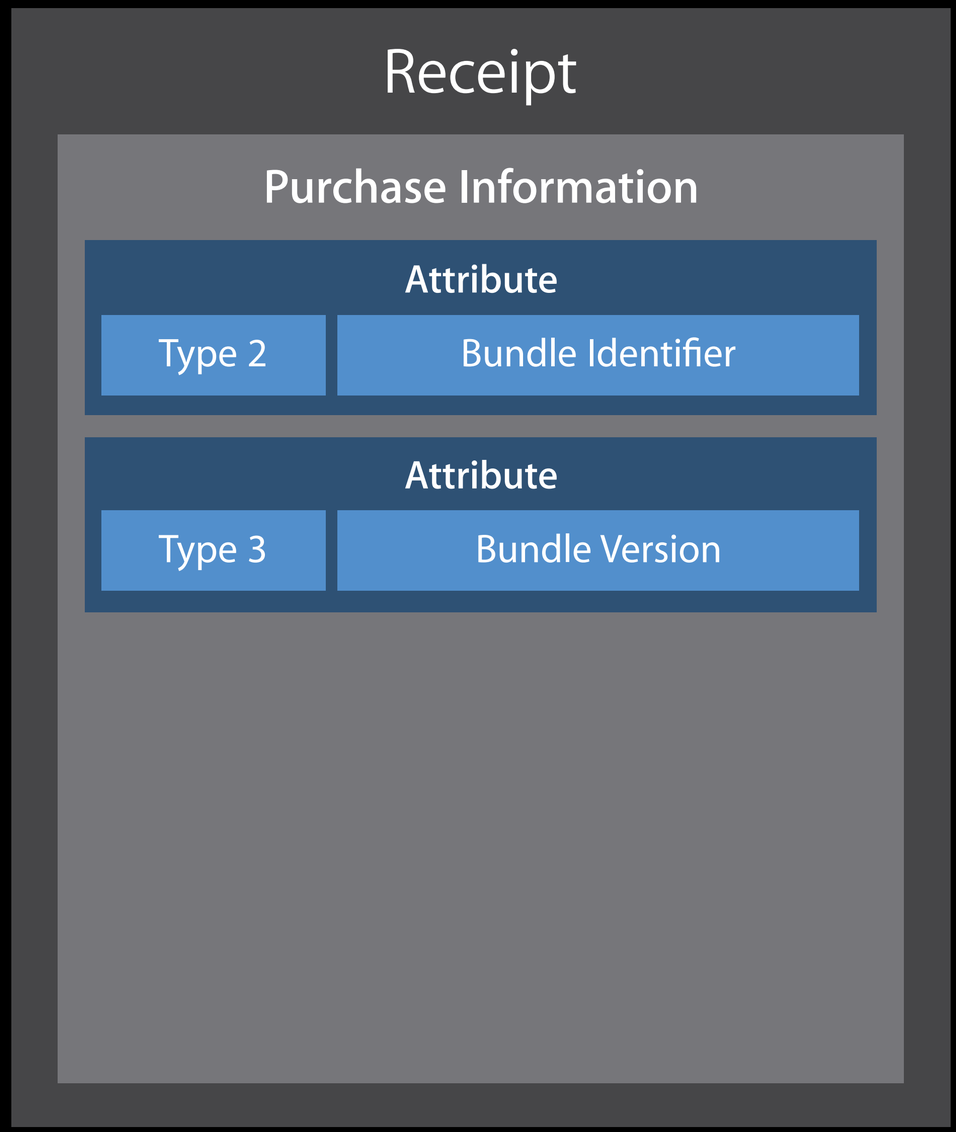
Verify Application

Check the Bundle Identifier

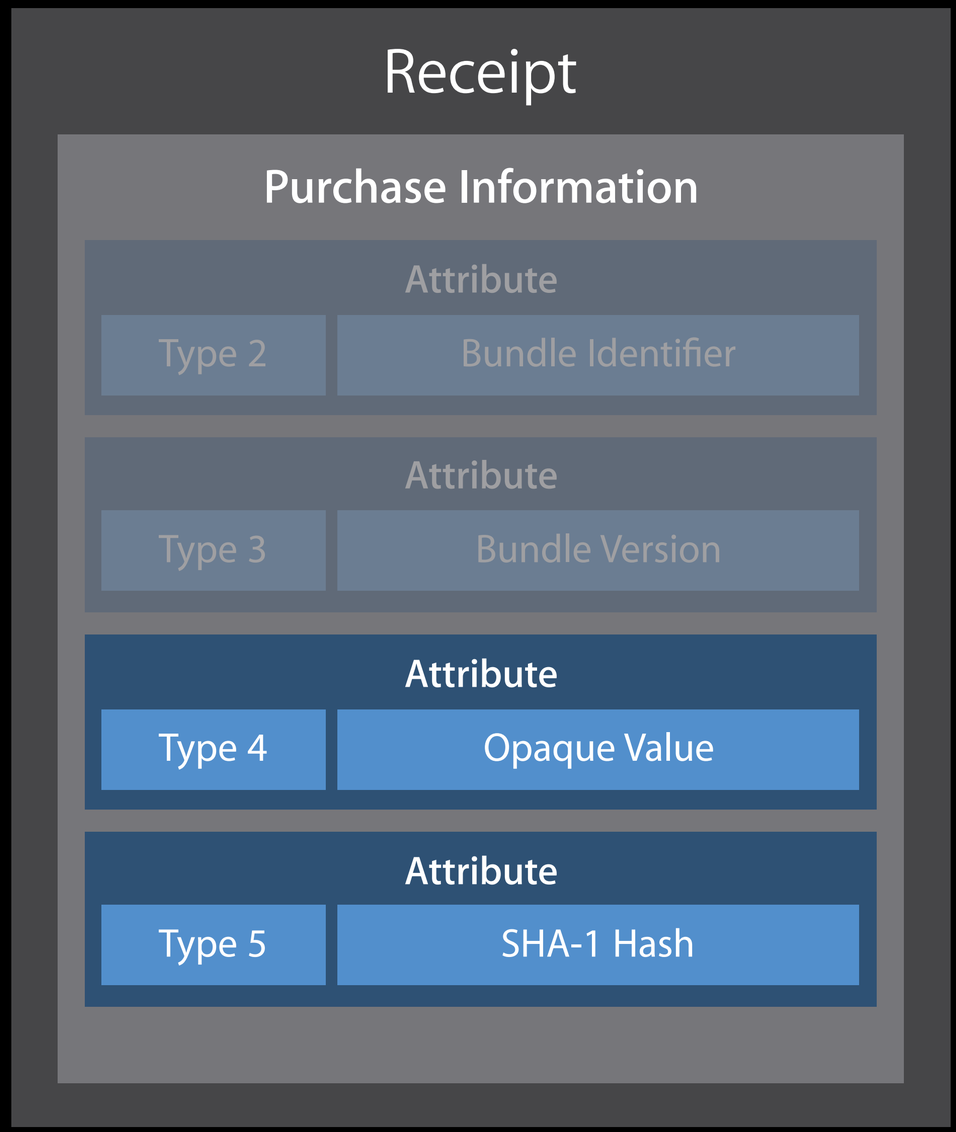
Check the Bundle Version

Use hardcoded values

- Not Info.plist values



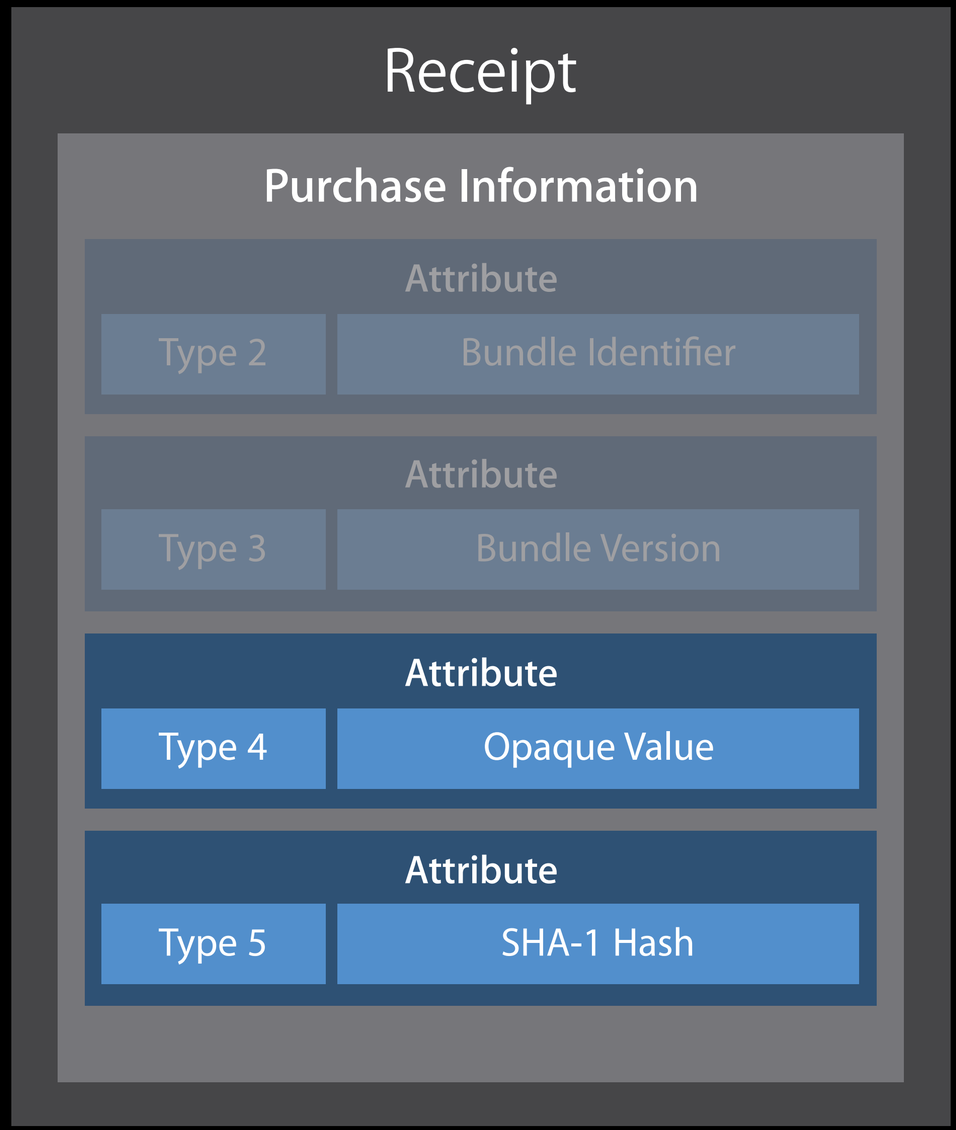
Verify Device



Verify Device

Attribute 5 is a SHA-1 hash of 3 key values

- Bundle ID
- Device Identifier
- Opaque Value

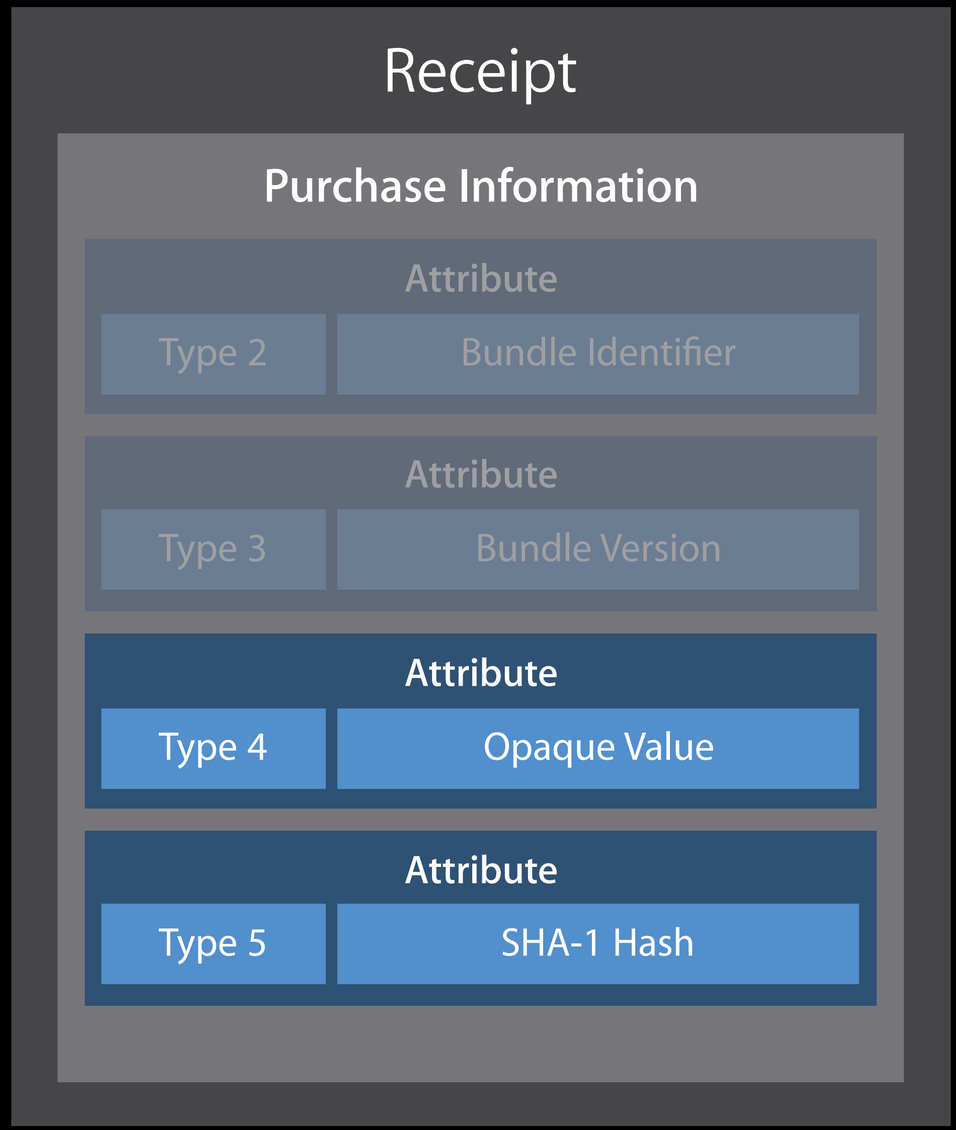


Verify Device

Attribute 5 is a SHA-1 hash of 3 key values

- Bundle ID
- Device Identifier
- Opaque Value

The App Store knows these at time of purchase



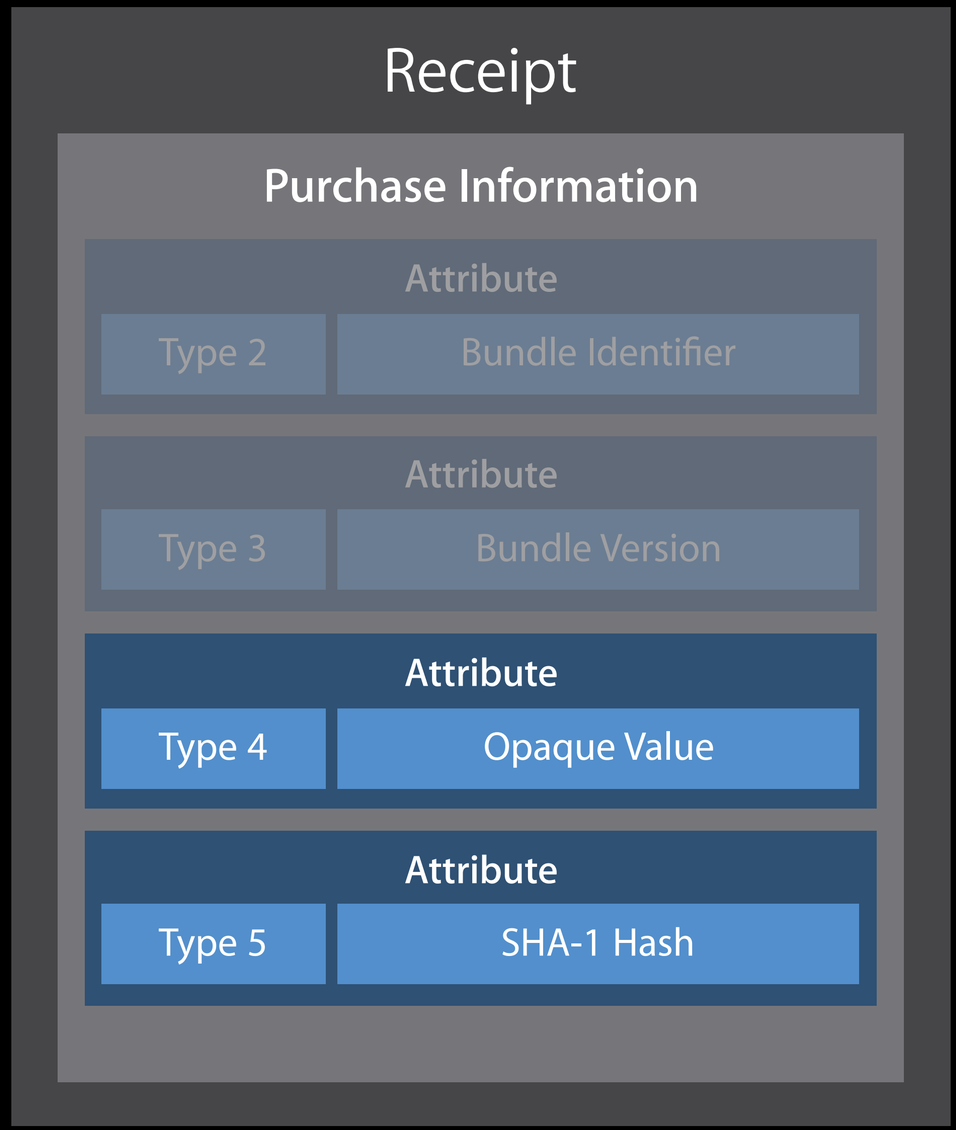
Verify Device

Attribute 5 is a SHA-1 hash of 3 key values

- Bundle ID
- Device Identifier
- Opaque Value

The App Store knows these at time of purchase

Your app knows them at time of verification



Verify Device

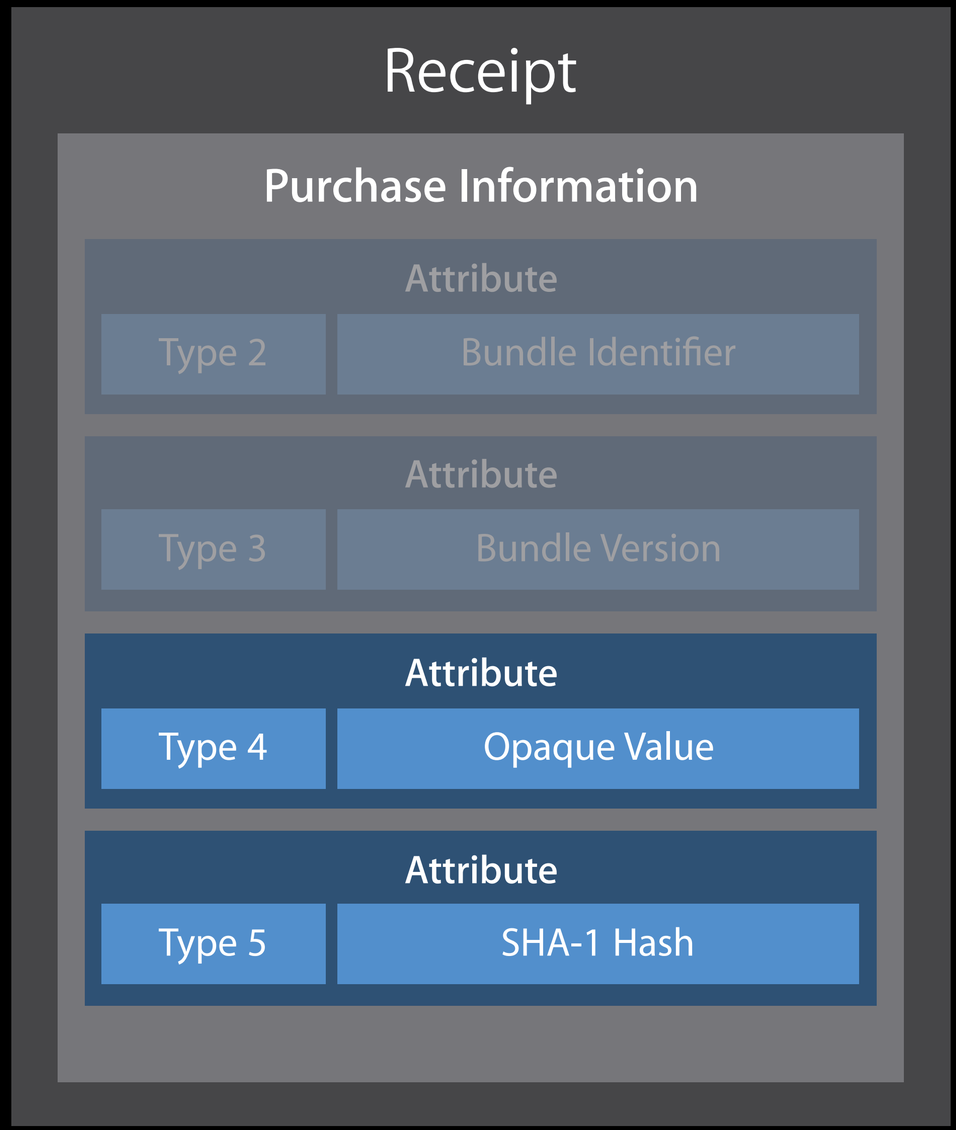
Attribute 5 is a SHA-1 hash of 3 key values

- Bundle ID
- Device Identifier
- Opaque Value

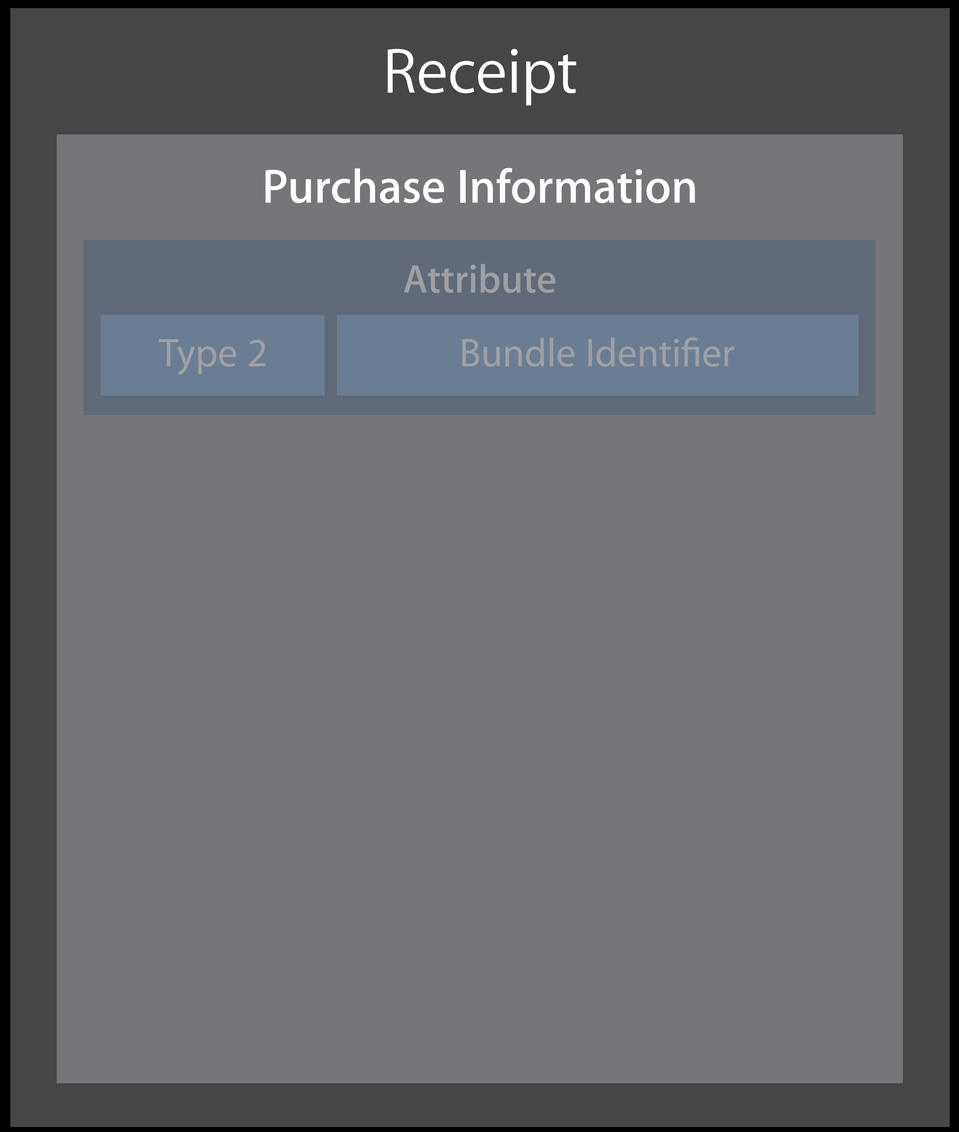
The App Store knows these at time of purchase

Your app knows them at time of verification

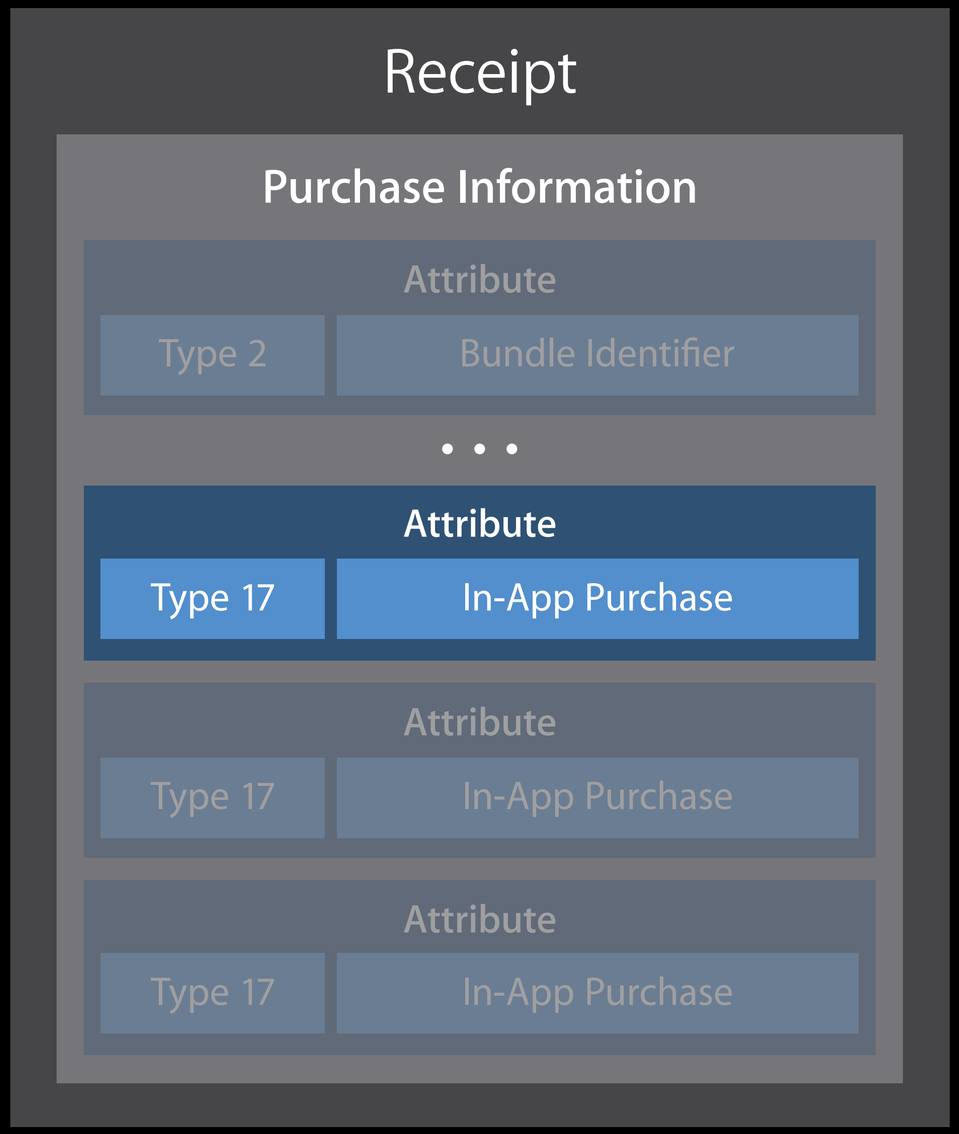
Unique to your app on this device



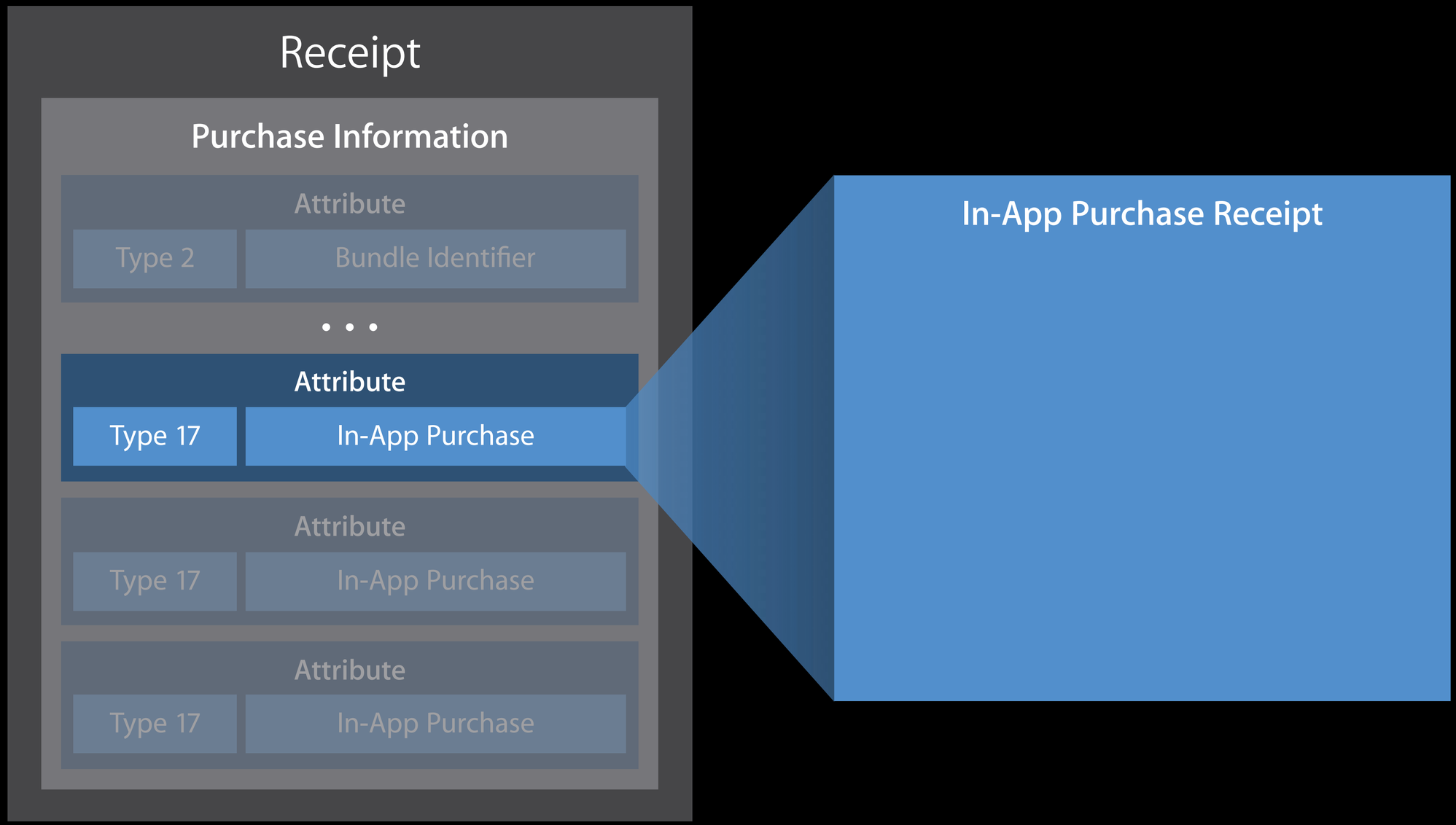
In-App Purchase Attributes



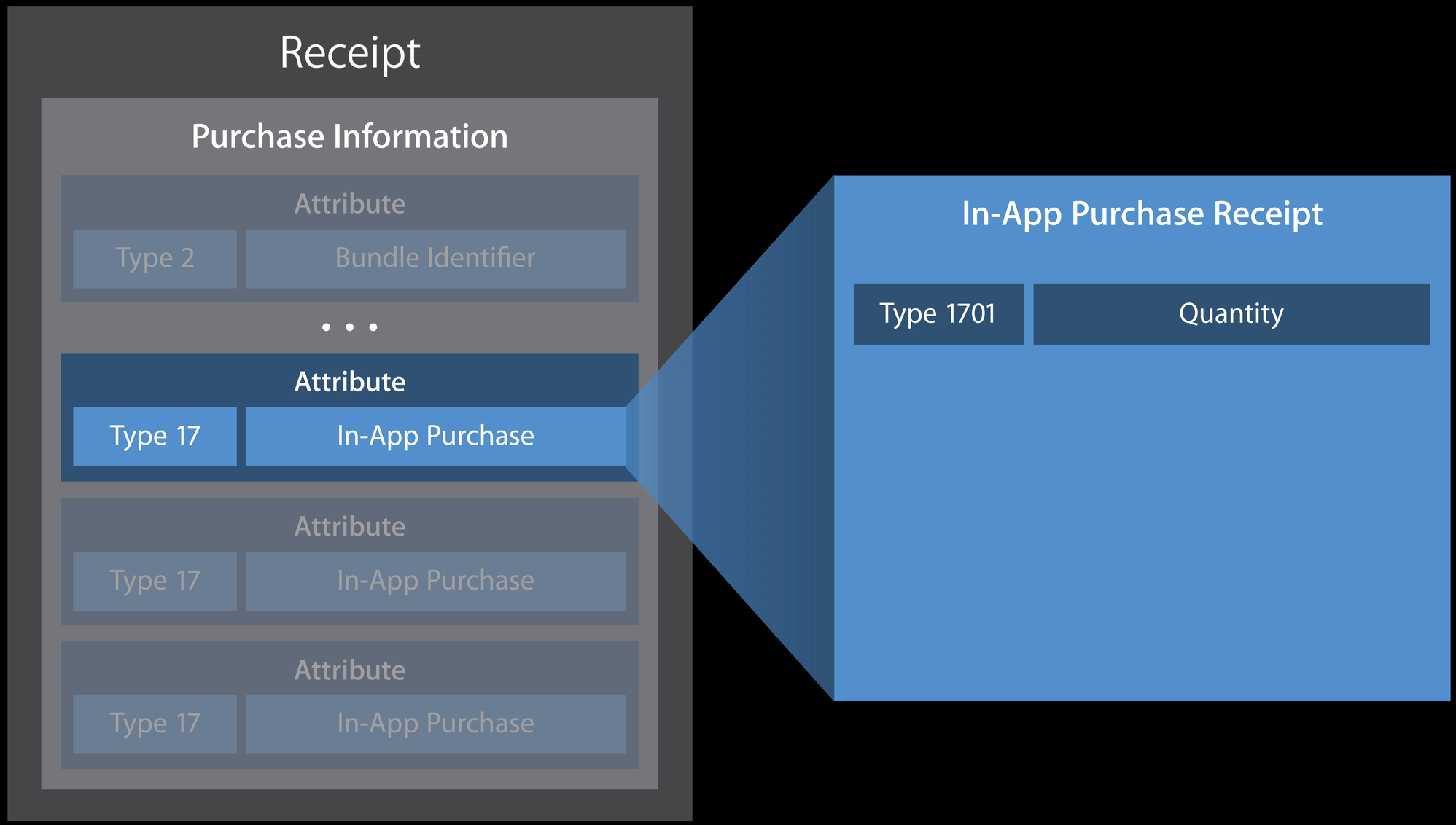
In-App Purchase Attributes



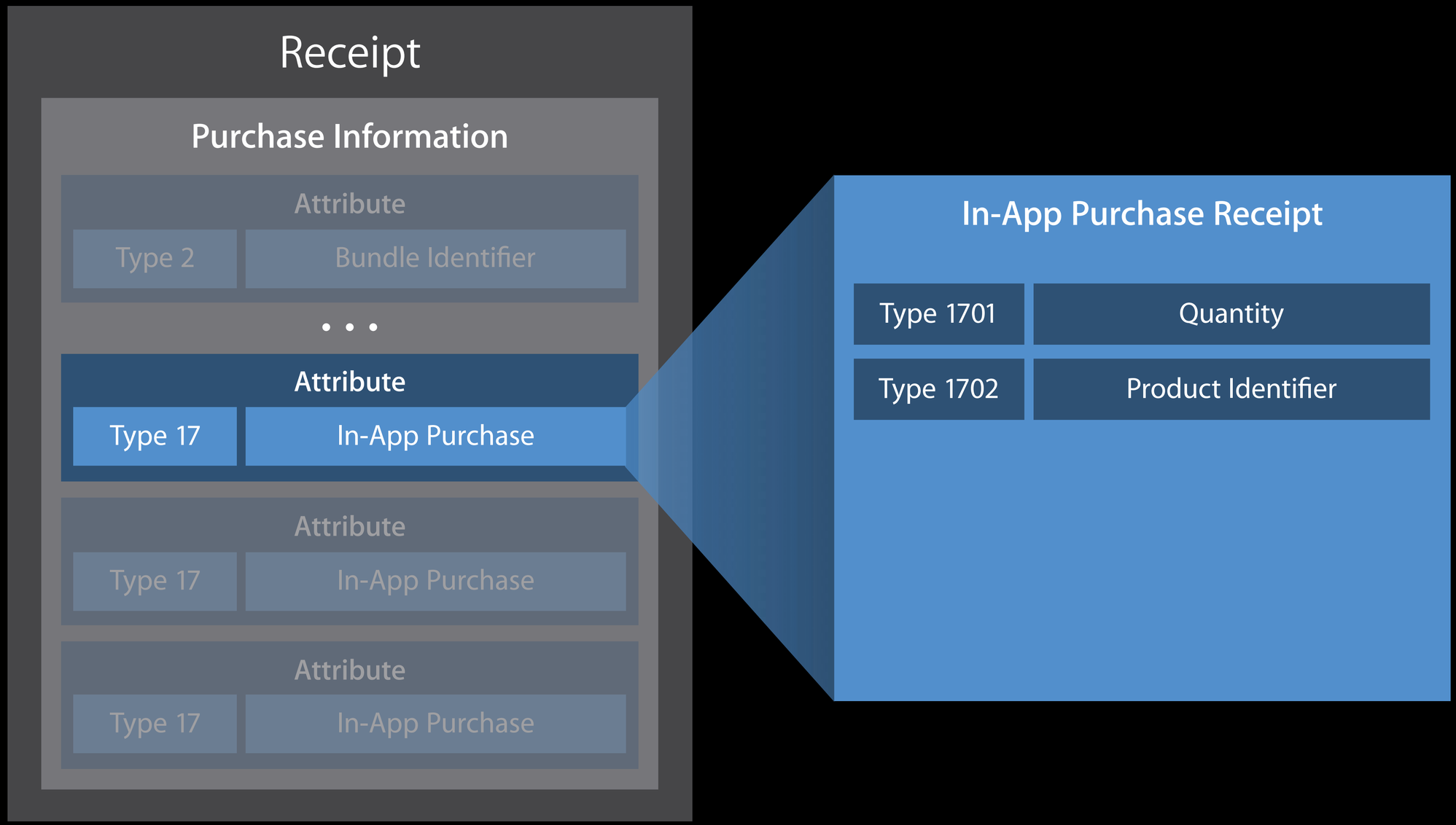
In-App Purchase Attributes



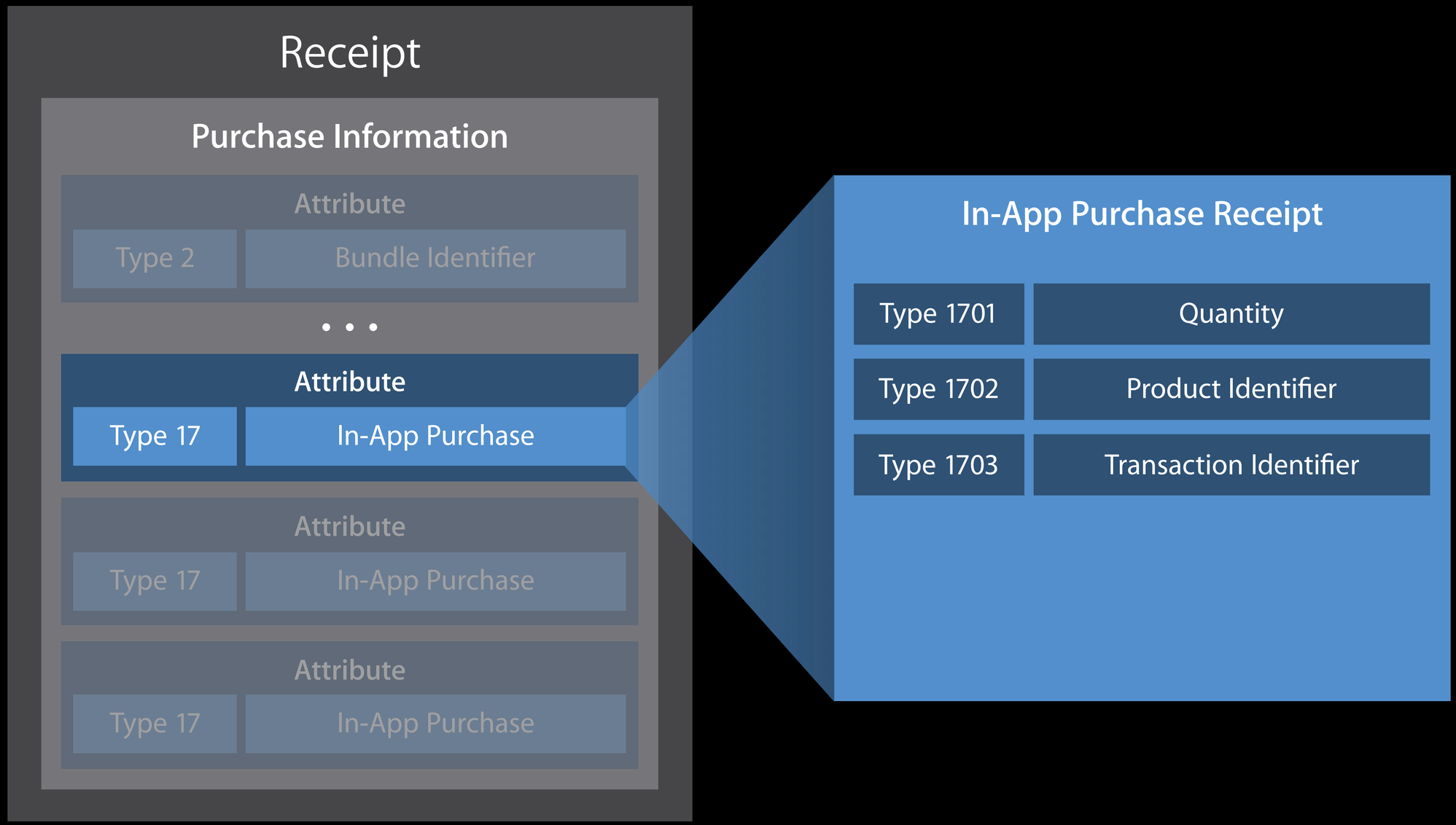
In-App Purchase Attributes



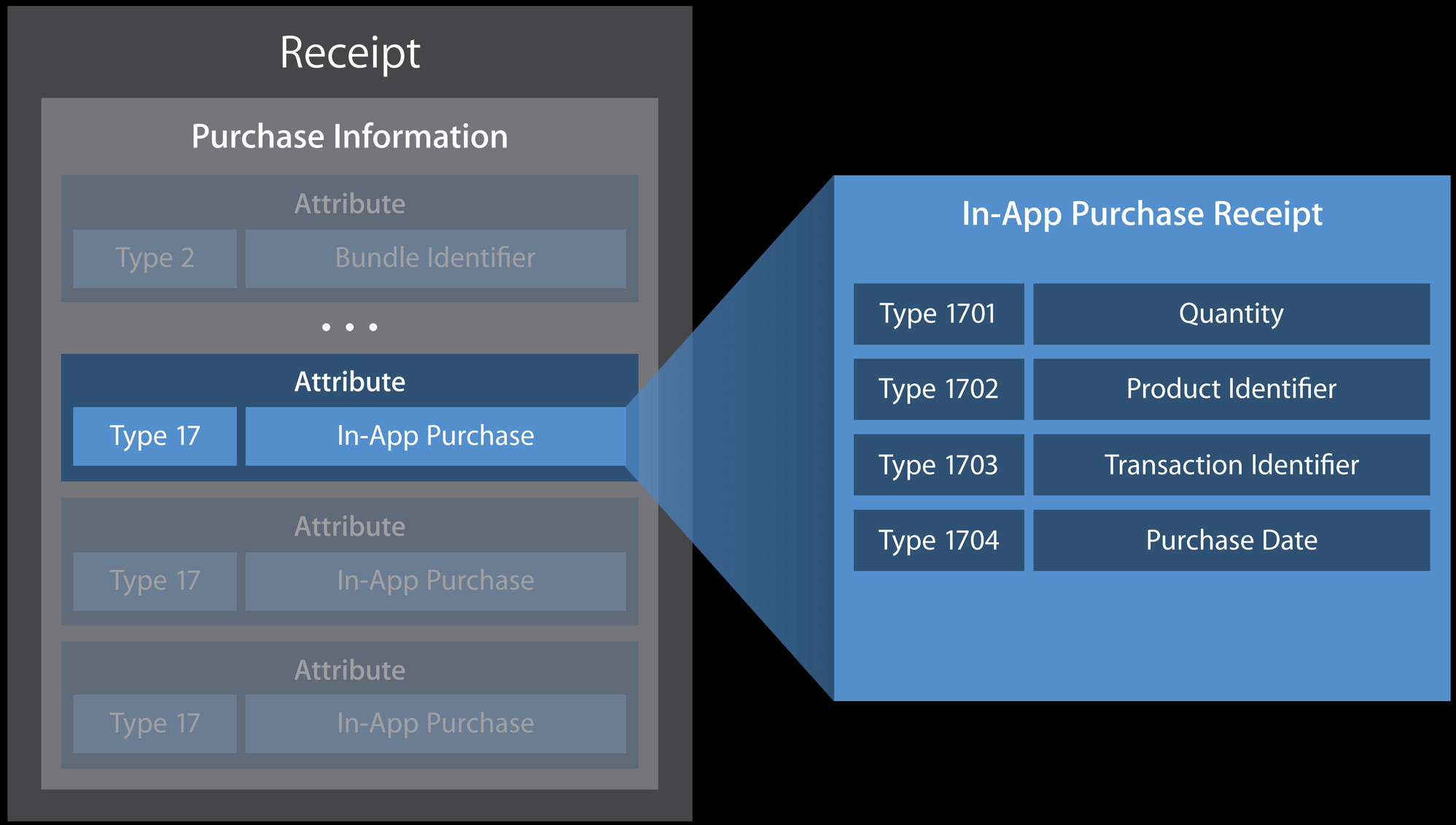
In-App Purchase Attributes



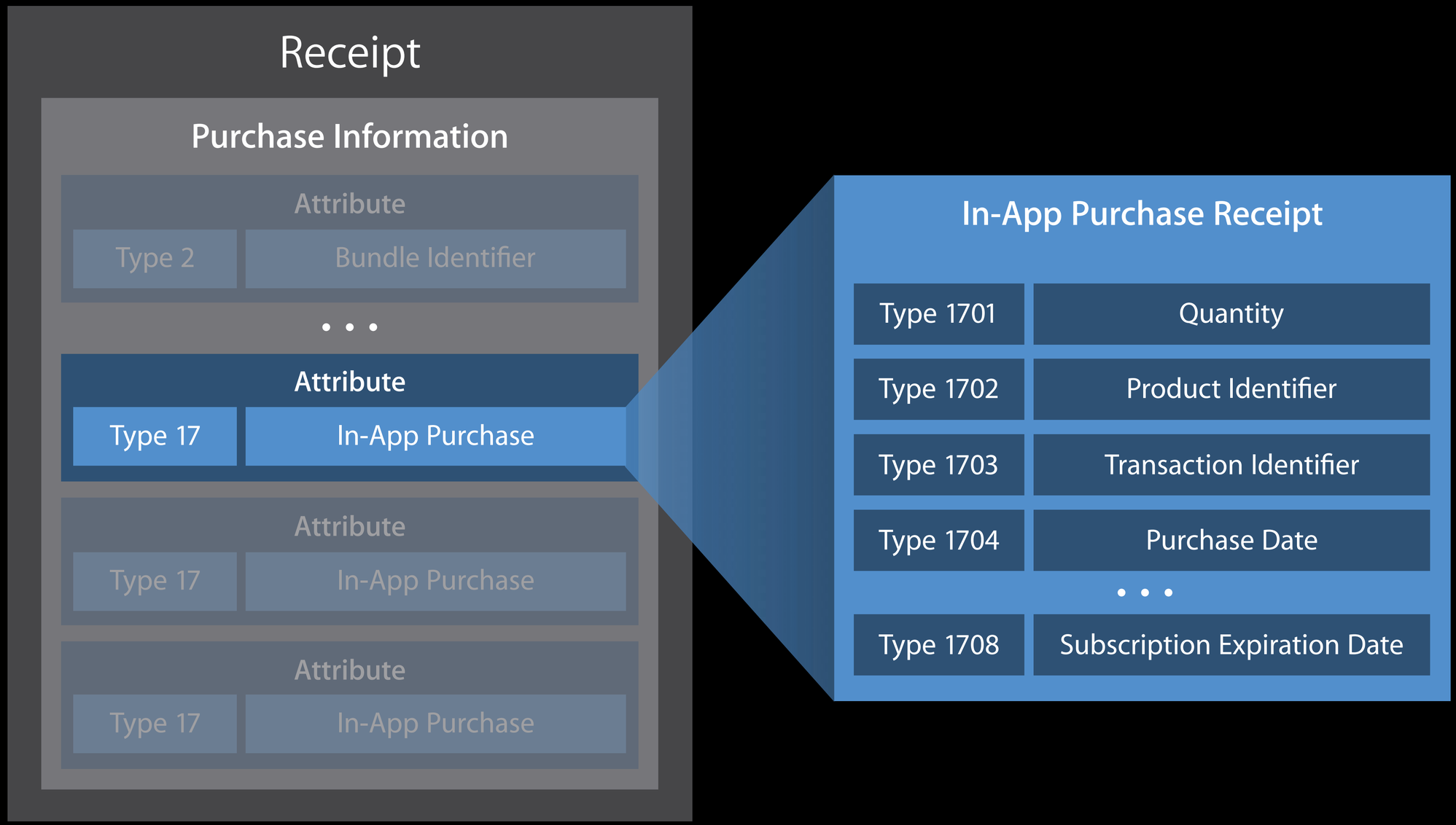
In-App Purchase Attributes



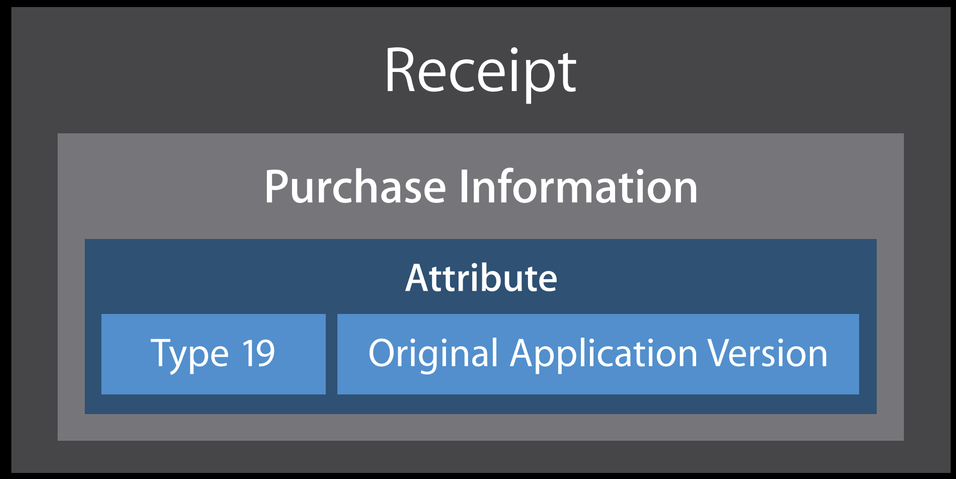
In-App Purchase Attributes



In-App Purchase Attributes

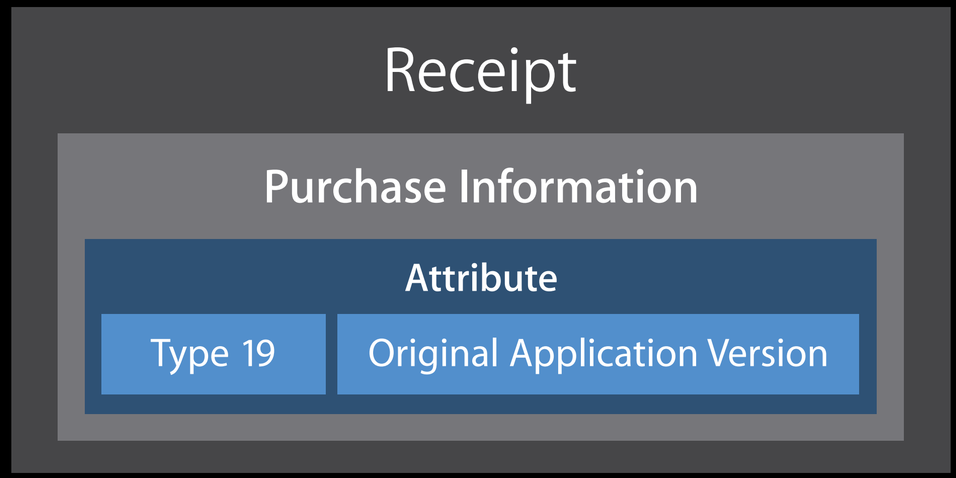


Switching to Subscriptions



Switching to Subscriptions

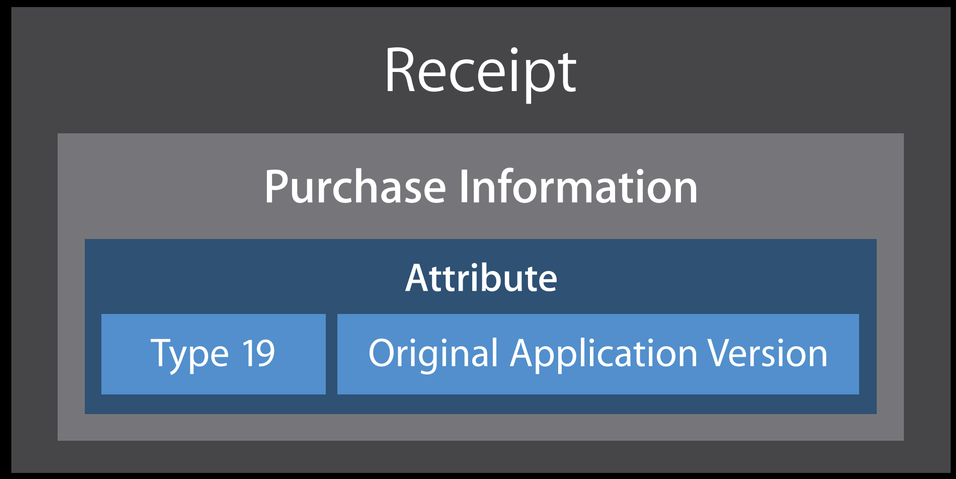
Original application version in the receipt



Switching to Subscriptions

Original application version in the receipt

Know whether to treat the app as the paid version, or the subscription version



Transaction Lifecycle

Transaction Lifecycle

Consumable and non-renewing subscriptions

- Will only appear once
- In the receipt issued at time of purchase
- Will not be present in subsequent receipts issued

Transaction Lifecycle

Consumable and non-renewing subscriptions

- Will only appear once
- In the receipt issued at time of purchase
- Will not be present in subsequent receipts issued

Non-consumable and auto-renewable subscriptions

- Always in the receipt
- Can be restored via StoreKit API

Receipt Refresh on iOS

Receipt Refresh on iOS

If the receipt doesn’t exist or is invalid, Refresh the receipt using StoreKit

Receipt Refresh on iOS

If the receipt doesn’t exist or is invalid, Refresh the receipt using StoreKit

Receipt refresh will require network

Receipt Refresh on iOS

If the receipt doesn’t exist or is invalid, Refresh the receipt using StoreKit

Receipt refresh will require network

Store sign-in will be required

Receipt Refresh on iOS

If the receipt doesn't exist or is invalid, Refresh the receipt using StoreKit

Receipt refresh will require network

Store sign-in will be required

Avoid continuous loop of validate-and-refresh

Receipt Refresh on iOS

If the receipt doesn't exist or is invalid, Refresh the receipt using StoreKit

Receipt refresh will require network

Store sign-in will be required

Avoid continuous loop of validate-and-refresh

```
let request = SKReceiptRefreshRequest()
request.delegate = self;
request.start()
```

Receipt Refresh on macOS

Receipt Refresh on macOS

If the receipt is invalid

Receipt Refresh on macOS

If the receipt is invalid

Exit with code 173 to refresh receipt

Receipt Refresh on macOS

If the receipt is invalid

Exit with code 173 to refresh receipt

Receipt refresh will require network

Receipt Refresh on macOS

If the receipt is invalid

Exit with code 173 to refresh receipt

Receipt refresh will require network

Store sign-in will be required

Receipt Refresh on macOS

If the receipt is invalid

Exit with code 173 to refresh receipt

Receipt refresh will require network

Store sign-in will be required

```
// Receipt is invalid
exit(173);
```

Server-to-Server Validation

Server-to-Server Validation

Allows your servers to validate the receipt before issuing content

Server-to-Server Validation

Allows your servers to validate the receipt before issuing content

Your app sends the receipt to your servers

- Your server sends the receipt to Apple's server
- Never send the receipt directly from your app to Apple's server

Server-to-Server Validation

Allows your servers to validate the receipt before issuing content

Your app sends the receipt to your servers

- Your server sends the receipt to Apple's server
- Never send the receipt directly from your app to Apple's server

Response is in JSON

In-App Purchase Process

Load In-App Identifiers	Fetch Product Info	Show In-App UI	Make Purchase	Process Transaction	Make Asset Available	Finish Transaction

Make Asset Available

Make Asset Available

Unlock functionality in your app

Make Asset Available

Unlock functionality in your app

Download additional content

On-Demand Resources

On-Demand Resources

Hosted on the App Store

On-Demand Resources

Hosted on the App Store

Can contain any data type except executable Swift, Objective-C, C, or C++ code

On-Demand Resources

Hosted on the App Store

Can contain any data type except executable Swift, Objective-C, C, or C++ code

Available on iOS and tvOS

On-Demand Resources

Hosted on the App Store

Can contain any data type except executable Swift, Objective-C, C, or C++ code

Available on iOS and tvOS

Optimizing On-Demand Resources Mission Thursday 10:00AM

Hosted In-App Purchase Content

Hosted In-App Purchase Content

Hosted on Apple's servers

Hosted In-App Purchase Content

Hosted on Apple's servers

Scalable and reliable

Hosted In-App Purchase Content

Hosted on Apple's servers

Scalable and reliable

Downloads in background

Hosted In-App Purchase Content

Hosted on Apple's servers

Scalable and reliable

Downloads in background

Up to 2GB per in-app purchasable product

Hosted In-App Purchase Content

Hosted on Apple's servers

Scalable and reliable

Downloads in background

Up to 2GB per in-app purchasable product

Supported on iOS, tvOS, and macOS

```
// Hosted Content

func paymentQueue(_ queue: SKPaymentQueue, updatedTransactions transactions
    [SKPaymentTransaction]) {
    for transaction in transactions {
        if transaction.downloads.count > 0 {
            SKPaymentQueue.defaultQueue().start(transaction.downloads)
        }
    }
}
```

```
    // Hosted Content

    func paymentQueue(_ queue: SKPaymentQueue, updatedTransactions transactions
        [SKPaymentTransaction]) {
        for transaction in transactions {
            if transaction.downloads.count > 0 {
                SKPaymentQueue.defaultQueue().start(transaction.downloads)
            }
        }
    }
```

```
// Hosted Content

func paymentQueue(_ queue: SKPaymentQueue, updatedTransactions transactions
    [SKPaymentTransaction]) {
    for transaction in transactions {
        if transaction.downloads.count > 0 {
            SKPaymentQueue.defaultQueue().start(transaction.downloads)
        }
    }
}
```

```
// Hosted Content

func paymentQueue(_ queue: SKPaymentQueue, updatedTransactions transactions
    [SKPaymentTransaction]) {
    for transaction in transactions {
        if transaction.downloads.count > 0 {
            SKPaymentQueue.defaultQueue().start(transaction.downloads)
        }
    }
}
```

```
// Hosted Content

func paymentQueue(_ queue: SKPaymentQueue, updatedTransactions transactions
    [SKPaymentTransaction]) {
    for transaction in transactions {
        if transaction.downloads.count > 0 {
            SKPaymentQueue.defaultQueue().start(transaction.downloads)
        }
    }
}
```

```swift
// Hosted Content

func paymentQueue(_ queue: SKPaymentQueue, updatedDownloads downloads: [SKDownload]) {
    for download in downloads {
        download.progress
        download.timeRemaining
        download.error

        if download.downloadState == .finished {
            download.contentURL
        }
    }
}
```

```
// Hosted Content

func paymentQueue(_ queue: SKPaymentQueue, updatedDownloads downloads: [SKDownload]) {
    for download in downloads {
        download.progress
        download.timeRemaining
        download.error

        if download.downloadState == .finished {
            download.contentURL
        }
    }
}
```

```
// Hosted Content

func paymentQueue(_ queue: SKPaymentQueue, updatedDownloads downloads: [SKDownload]) {
    for download in downloads {
        download.progress
        download.timeRemaining
        download.error

        if download.downloadState == .finished {
            download.contentURL
        }
    }
}
```

```swift
// Hosted Content

func paymentQueue(_ queue: SKPaymentQueue, updatedDownloads downloads: [SKDownload]) {
    for download in downloads {
        download.progress
        download.timeRemaining
        download.error

        if download.downloadState == .finished {
            download.contentURL
        }
    }
}
```

```swift
// Hosted Content

func paymentQueue(_ queue: SKPaymentQueue, updatedDownloads downloads: [SKDownload]) {
    for download in downloads {
        download.progress
        download.timeRemaining
        download.error

        if download.downloadState == .finished {
            download.contentURL
        }
    }
}
```

Self-Hosted Content

Self-Hosted Content

Use background download APIs

Self-Hosted Content

Use background download APIs

- Content is downloaded even when your app is not active

Self-Hosted Content

Use background download APIs

- Content is downloaded even when your app is not active
- NSURLConnection is deprecated

```
// Self-Hosted Content

let config = NSURLSessionConfiguration.backgroundSessionConfiguration(withIdentifier:
    "MyBackgroundSession")
let session = NSURLSession(configuration: config, delegate: self, delegateQueue: queue)
let request = NSURLRequest(url: myURL)
let downloadTask = session.downloadTask(with: request)
downloadTask.resume()
```

```
// Self-Hosted Content

let config = NSURLSessionConfiguration.backgroundSessionConfiguration(withIdentifier:
    "MyBackgroundSession")
let session = NSURLSession(configuration: config, delegate: self, delegateQueue: queue)
let request = NSURLRequest(url: myURL)
let downloadTask = session.downloadTask(with: request)
downloadTask.resume()
```

```
// Self-Hosted Content

let config = NSURLSessionConfiguration.backgroundSessionConfiguration(withIdentifier:
    "MyBackgroundSession")
let session = NSURLSession(configuration: config, delegate: self, delegateQueue: queue)
let request = NSURLRequest(url: myURL)
let downloadTask = session.downloadTask(with: request)
downloadTask.resume()
```

```
// Self-Hosted Content

let config = NSURLSessionConfiguration.backgroundSessionConfiguration(withIdentifier:
    "MyBackgroundSession")
let session = NSURLSession(configuration: config, delegate: self, delegateQueue: queue)
let request = NSURLRequest(url: myURL)
let downloadTask = session.downloadTask(with: request)
downloadTask.resume()
```

```
// Self-Hosted Content

let config = NSURLSessionConfiguration.backgroundSessionConfiguration(withIdentifier:
    "MyBackgroundSession")
let session = NSURLSession(configuration: config, delegate: self, delegateQueue: queue)
let request = NSURLRequest(url: myURL)
let downloadTask = session.downloadTask(with: request)
downloadTask.resume()
```

```
// Self-Hosted Content

func urlSession(_ session: NSURLSession,
                downloadTask: NSURLSessionDownloadTask,
                didWriteData bytesWritten: Int64,
                totalBytesWritten: Int64,
                totalBytesExpectedToWrite: Int64) {
    // Do something with progress
}
```

```
// Self-Hosted Content

func urlSession(_ session: NSURLSession,
                downloadTask: NSURLSessionDownloadTask,
                didWriteData bytesWritten: Int64,
                totalBytesWritten: Int64,
                totalBytesExpectedToWrite: Int64) {
    // Do something with progress
}
```

```
// Self-Hosted Content

func application(_ application: UIApplication,
                 handleEventsForBackgroundURLSession identifier: String,
                 completionHandler: () -> Void) {
   let config = NSURLSessionConfiguration.
        backgroundSessionConfiguration(withIdentifier: identifier)
    let session = NSURLSession(configuration: config, delegate: self, delegateQueue: queue)
    self.completionHandler = completionHandler // call when done
}
```

```
// Self-Hosted Content

func application(_ application: UIApplication,
                 handleEventsForBackgroundURLSession identifier: String,
                 completionHandler: () -> Void) {
    let config = NSURLSessionConfiguration.
        backgroundSessionConfiguration(withIdentifier: identifier)
     let session = NSURLSession(configuration: config, delegate: self, delegateQueue: queue)
     self.completionHandler = completionHandler // call when done
}
```

```
// Self-Hosted Content

func application(_ application: UIApplication,
                 handleEventsForBackgroundURLSession identifier: String,
                 completionHandler: () -> Void) {
   let config = NSURLSessionConfiguration.
        backgroundSessionConfiguration(withIdentifier: identifier)
    let session = NSURLSession(configuration: config, delegate: self, delegateQueue: queue)
    self.completionHandler = completionHandler // call when done
}
```

```
// Self-Hosted Content

func application(_ application: UIApplication,
                 handleEventsForBackgroundURLSession identifier: String,
                 completionHandler: () -> Void) {
   let config = NSURLSessionConfiguration.
        backgroundSessionConfiguration(withIdentifier: identifier)
    let session = NSURLSession(configuration: config, delegate: self, delegateQueue: queue)
    self.completionHandler = completionHandler // call when done
}
```

```
// Self-Hosted Content

func application(_ application: UIApplication,
                 handleEventsForBackgroundURLSession identifier: String,
                 completionHandler: () -> Void) {
   let config = NSURLSessionConfiguration.
        backgroundSessionConfiguration(withIdentifier: identifier)
    let session = NSURLSession(configuration: config, delegate: self, delegateQueue: queue)
    self.completionHandler = completionHandler // call when done
}
```

In-App Purchase Process

Load In-App Identifiers	Fetch Product Info	Show In-App UI	Make Purchase	Process Transaction	Make Asset Available	Finish Transaction

Finish the Transaction

Finish the Transaction

When the content is downloaded, finish the transaction

Finish the Transaction

When the content is downloaded, finish the transaction

- Otherwise, the payment will stay in the queue

Finish the Transaction

When the content is downloaded, finish the transaction

- Otherwise, the payment will stay in the queue
- If downloading Apple-hosted content, wait until after the download completes

Finish the Transaction

When the content is downloaded, finish the transaction

- Otherwise, the payment will stay in the queue
- If downloading Apple-hosted content, wait until after the download completes

```
SKPaymentQueue.defaultQueue().finishTransaction(transaction)
```

Restore Completed Transactions

Restore Completed Transactions

Restoring transactions allows the user to restore

- Non-consumable in-app purchases
- Auto-renewing subscriptions

Restore Completed Transactions

Restoring transactions allows the user to restore

- Non-consumable in-app purchases
- Auto-renewing subscriptions

Consumables and non-renewable subscriptions

- You must persist the state!

Restore Completed Transactions

Restore Completed Transactions

```
SKPaymentQueue.defaultQueue().restoreCompletedTransactions()
```

Restore Completed Transactions

```
SKPaymentQueue.defaultQueue().restoreCompletedTransactions()
```

Observe the queue

Restore Completed Transactions

```
SKPaymentQueue.defaultQueue().restoreCompletedTransactions()
```

Observe the queue

```
func paymentQueueRestoreCompletedTransactionsFinished(_ queue: SKPaymentQueue) {…}

func paymentQueue(_ queue: SKPaymentQueue,
    restoreCompletedTransactionsFailedWithError error: NSError) {…}
```

Restore Completed Transactions

```
SKPaymentQueue.defaultQueue().restoreCompletedTransactions()
```

Observe the queue

```
func paymentQueueRestoreCompletedTransactionsFinished(_ queue: SKPaymentQueue) {…}

func paymentQueue(_ queue: SKPaymentQueue,
    restoreCompletedTransactionsFailedWithError error: NSError) {…}
```

Inspect the receipt and unlock content and features accordingly

Tips for Passing App Review

Restore Button

Restore Button

You must have a Restore button

Restore Button

You must have a Restore button

Should be used only for

- Non-consumables
- Auto-renewable subscriptions

Restore Button

You must have a Restore button

Should be used only for

- Non-consumables
- Auto-renewable subscriptions

Restore and Purchase should be separate buttons

Auto-Renewable Subscriptions

Auto-Renewable Subscriptions

You must indicate a privacy policy URL

Auto-Renewable Subscriptions

You must indicate a privacy policy URL

Auto-renewable subscription must be in marketing text

Auto-Renewable Subscriptions

You must indicate a privacy policy URL

Auto-renewable subscription must be in marketing text

After subscribing, the latest issue must become downloadable

Auto-Renewable Subscriptions

You must indicate a privacy policy URL

Auto-renewable subscription must be in marketing text

After subscribing, the latest issue must become downloadable

Paid subscription must provide non-free content

Non-Renewing Subscriptions

Non-Renewing Subscriptions

Asking users to register should be optional

- Unless you offer account-based features

Purchases

Purchases

Purchases must work!

Summary

Summary

Always observe the Payment Queue

Summary

Always observe the Payment Queue

Fetch localized product information from the App Store

Summary

Always observe the Payment Queue

Fetch localized product information from the App Store

Display pricing using the product's price locale

Summary

Always observe the Payment Queue

Fetch localized product information from the App Store

Display pricing using the product's price locale

Use the receipt to validate your purchases

Summary

Always observe the Payment Queue

Fetch localized product information from the App Store

Display pricing using the product's price locale

Use the receipt to validate your purchases

Make the content available

Summary

Always observe the Payment Queue

Fetch localized product information from the App Store

Display pricing using the product's price locale

Use the receipt to validate your purchases

Make the content available

Finish the transaction

Summary

Always observe the Payment Queue

Fetch localized product information from the App Store

Display pricing using the product's price locale

Use the receipt to validate your purchases

Make the content available

Finish the transaction

Allow the user to restore complete transactions

More Information

https://developer.apple.com/wwdc16/702

Related Sessions

Introducing Expanded Subscriptions in iTunes Connect	Pacific Heights	Tuesday 4:00PM
Optimizing On-Demand Resources	Mission	Thursday 10:00AM

Labs

In-App Purchase/Subscriptions Lab 1	Frameworks Lab B	Wednesday 9:00AM
In-App Purchase/Subscriptions Lab 2	Graphics, Games, and Media Lab A	Friday 9:00AM

WWDC16