JUNE 19, 2018

UNDER THE HOOD: CODE SIGNING ON IOS

BY PATRICK BUTKIEWICZ

Accenture Interactive

AGENDA

- Introduction
- Introduction to Code Signing
- Code Signing on iOS
- Common Issues
- Closing

01 INTRODUCTION

PATRICK BUTKIEWICZ

Senior Principle Software Engineer,
Intrepid Pursuits / Accenture Interactive

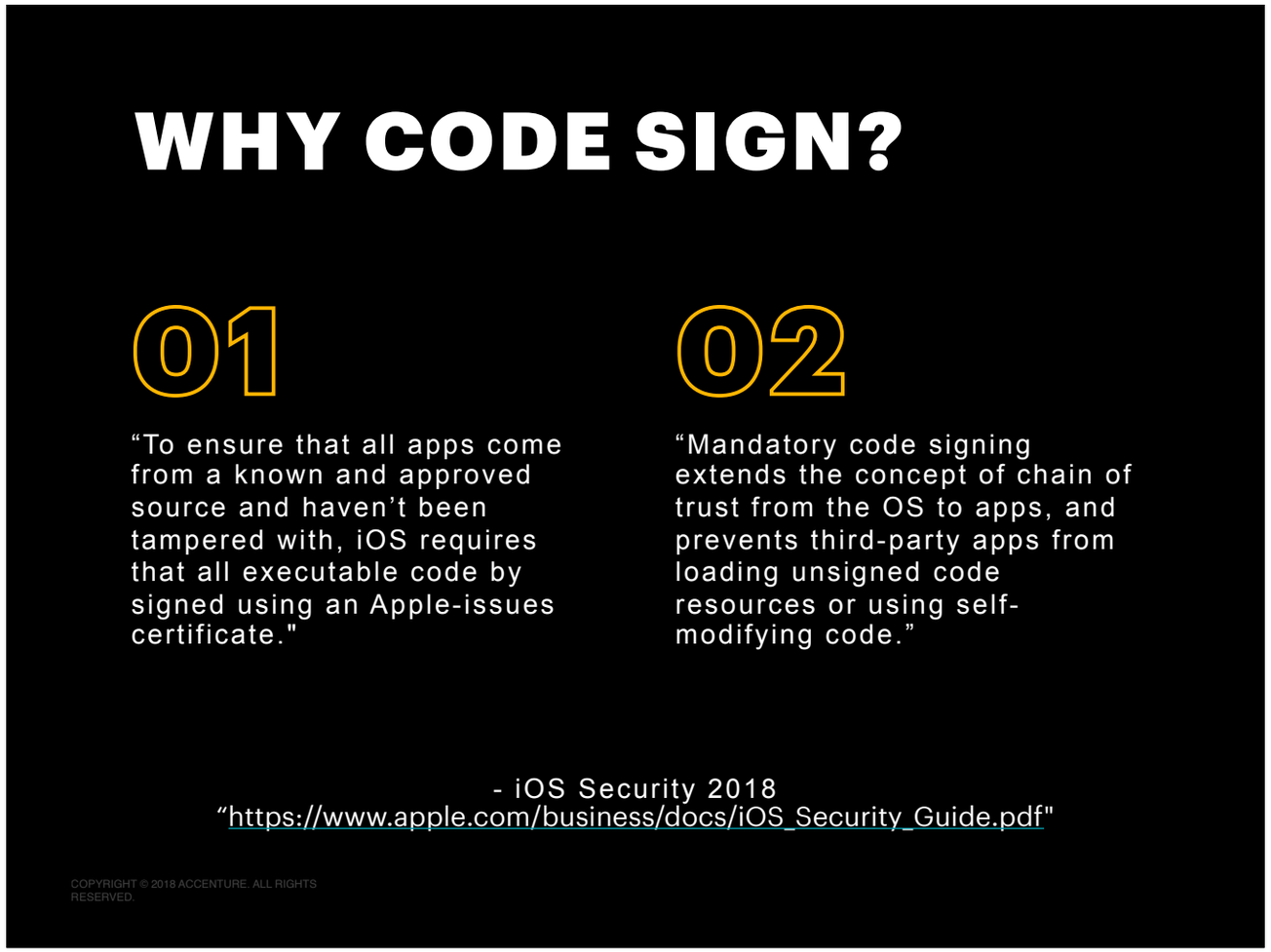
@patbutkiewicz
pat.butkiewicz@accenture.com

- Worked as an iOS developer since 2012
- Developed the CI/CD pipeline for the iOS team at Intrepid
- Manage our CI/CD infrastructure, perform various devops tasks, and work as a cloud architect on AWS
- Not: A cryptographer.
- Knowledge from years of working with our CI environment, troubleshooting issues within the iOS ecosystem, as well as some independent crypto research

02

INTRODUCTION TO CODE SIGNING

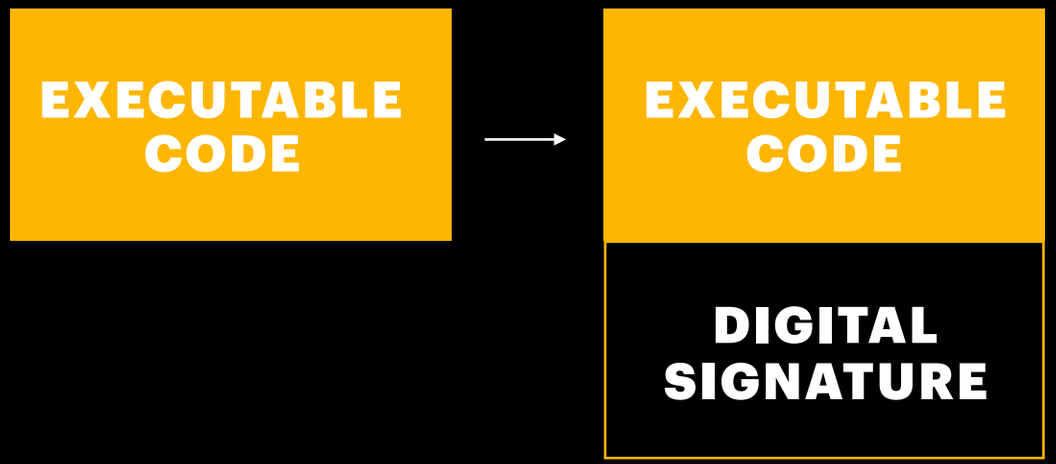
- **The Question:** Why is Code Signing a necessity in the first place?



- **WHY code sign?**
  - We want to trust that code submitted by a user hasn’t been tampered with.
    - We - Apple AND We - The end device user
  - An app that’s been code signed cannot be altered or else the system will **refuse** to run it.
  - **Question:** How is an application’s code signed

Pat: These sound like great features, how do I sign my own application?

HOW?



- Add a **Digital Signature**
  - True for ANY system implementing code signing
  - **Pat: What's a digital signature? And how does it verify the properties we seek from a signed application?**
    - To answer this we need to look at two fundamental concepts in cryptography

HASHING

A function that converts a long string of bytes into numbers

- **One-Way** function. No way to go backwards.
- Strong vs Weak hashing functions
  - Weak — Good chance of different strings of bytes being converted to the same number
  - Strong — You can't get the same number
  - You'll become famous for breaking strong functions

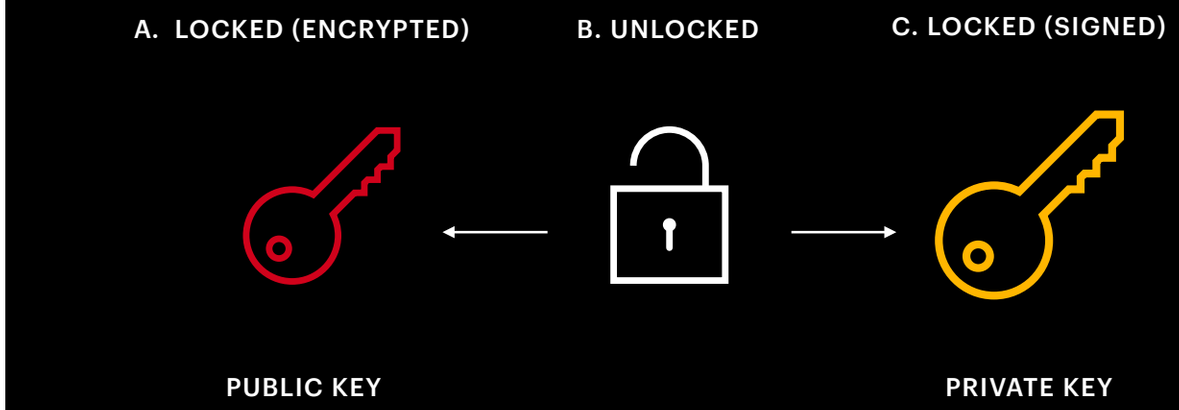
- This is an example of the MD5 hash function.
  - Even the slightest change (a capital letter) will produce a radically different hash
  - Not secure anymore
  - Apple uses the SHA256 hashing function

INTRODUCTION TO CODE SIGNING

PUBLIC-KEY CRYPTOGRAPHY

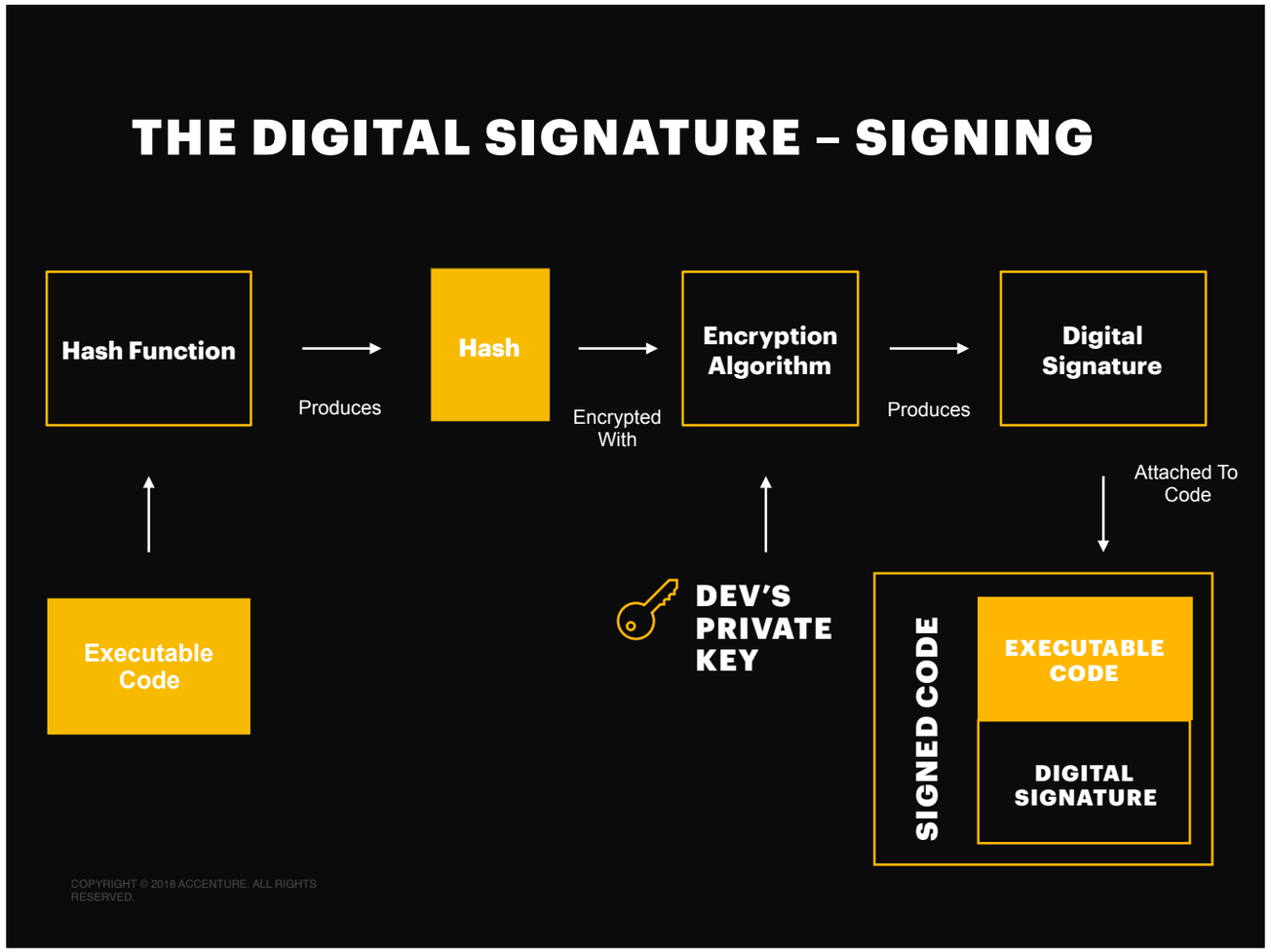
Asymmetric Encryption

Public Key and Private Key

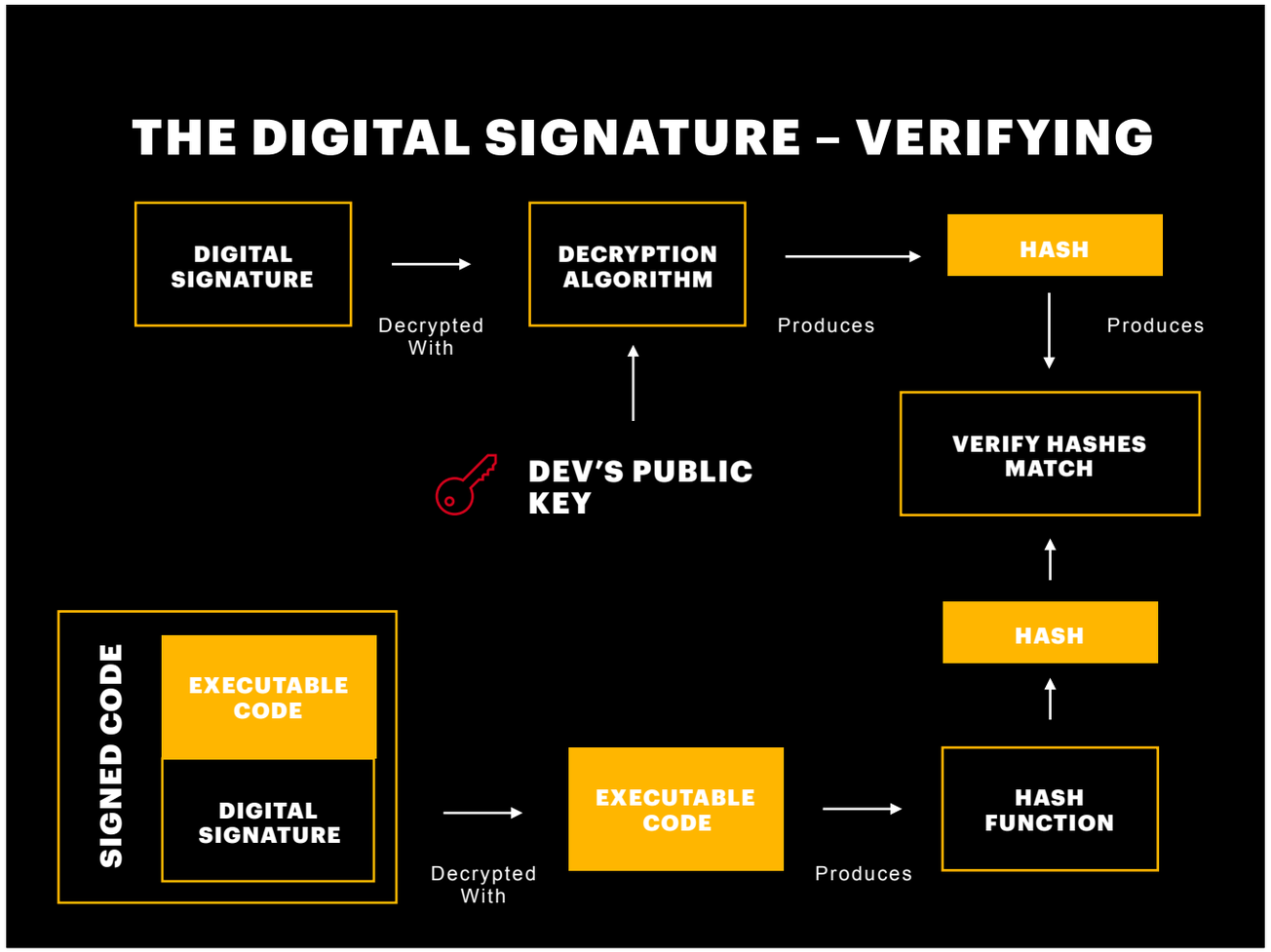


- **Symmetric Encryption:** One-key. Lock the box with the key. Need a copy of the key to unlock it

- **Asymmetric Encryption:** TWO-Keys
  - One **private** key, kept secret by the user always.
    - 3-State-Lock: Can ONLY turn the lock to the right
      - Guarantees the user is the only person who could have turned the lock that way, meaning the user is the **only** person who could have placed the contents in the “locked box”. The contents 100% came from this user.
  - A **public** key, of which many copies are made and freely given out.
    - 3-State-Lock: Can ONLY turn the lock to the left
      - Guarantees that only the user can unlock the contents. Anyone can encrypt some text with the user’s public key, and it can ONLY be decrypted by the user with the private key



1. The executable code is run through a hashing function, producing a unique hash for that code.
2. The hash is encrypted using the developer's private key, producing a *digital signature*.
3. The digital signature is attached to the code (potentially with the user's public key, and other information)
4. The application is now **Signed**



1. The signed code is split up into it's digital signature, and the original executable code
2. The original code is run through the same hash function as during signing
3. The developer's PUBLIC key is used to decrypt the digital signature.
4. The hash from the digital signature, and the computed hash are compared to verify authenticity
5. The program is run by the OS

03 CODE SIGNING ON IOS

THE APPLICATION SANDBOX

- Restricts access to system resources
- Works alongside code signing infrastructure
- App specifies the system resources it wants to use with **Entitlements**

- Provides a gateway to system resources.
- It’s what protects a malicious app from reading your contacts without permission, etc

MEMBER CENTER & KEYCHAINS

MEMBER CENTER

- Where you create and manage App IDs, provisioning profiles, certificates, devices.
  - Connected directly with Xcode

KEYCHAIN

- Local storage for certificates, profiles, keys, etc.

Keychain
- A keychain is actually a file you can export, and import to other machines.
- Access provided by the Keychain-Access application on macs.

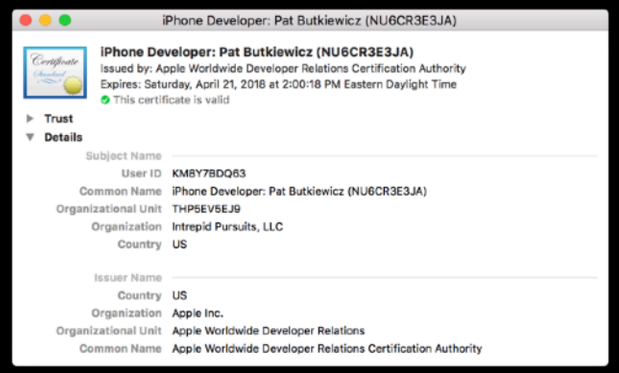
CERTIFICATE SIGNING REQUEST

- Request a certificate from a Certificate Authority (Apple), who can then verify the details, and issue the developer a certificate.
- Generated from Keychain-Access or CLI.
- Utilizes public-key infrastructure. Generates a private/public key-pair.
- Resulting CSR is uploaded to Apple, and a certificate is issued

- Message sent from an applicant to a Certificate Authority (CA) to apply for a digital certificate.
  - Applicable to ALL platforms that support codesigning.
  - Typical for SSL certificates on web.
- STEPS:
  - While creating CSR, the public/private key pair is generated under the hood.
  - The public key is attached to your CSR
  - The private key is kept inside your local machine.
  - CSR is uploaded to apple.

CERTIFICATES

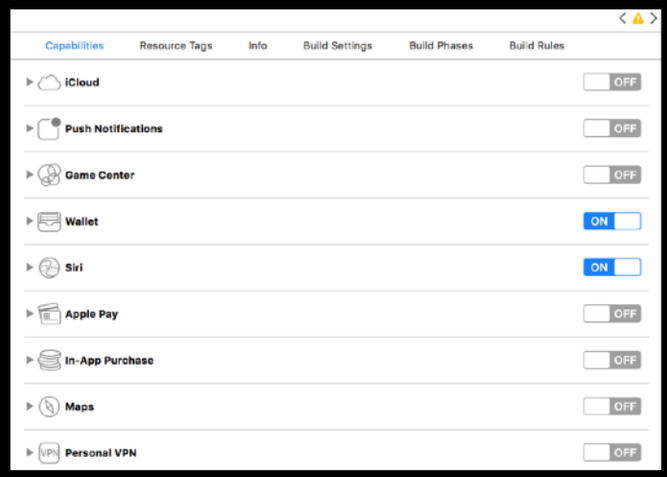
- Public key, combined with additional information and signed by a CA (Apple)
- Developer & Distribution
- Combined with a private key forms a “Code Signing Identity”
- X.509 Standard



- Very broadly a certificate is a public key with some extra information which itself has been signed by some authority. In this case the authority is Apple.
- The signature expires at some point, so anyone checking it must have a working clock.
- Two certificates are generated usually, developer and distribution. Certs are prefixed with respective names, but this is codified in the certificate in the “extension entries”. Apple Developer Certificate (Submission), or Apple Developer Certificate (Development)
  - **Developer Certificate**: Used to build apps for your own devices
  - **Distribution Certificate:** Used for submission to the app store, or as an enterprise app\
- **OTHER INFO IN CERT:**
  - Issuer
  - Public Key info, such as which algorithm was used

ENTITLEMENTS

- Specify which resources of the system an app is allowed to use, and under what circumstances.
- An XML file is generated by editing the “capabilities” tab in Xcode.



- XML File generated by Xcode and passed to the CodeSign application.
- Certain rules in place around certain entitlements
  - e.g. entitlement to attach a debugger only allowed when signing with a Developer certificate

PROVISIONING PROFILES

- Bind certificates, the sandbox, and code signing together.
- A container for the information needed by the OS to determine if an app should be allowed to run.
- Development, Ad-hoc, Enterprise, App-Store
- Signed by Apple

- Glue that binds all of the sandbox and code signing infrastructure together
- Determines if a particular app can run on a particular device
- Enables such things as debugging on development devices, to large scale ad-hoc and enterprise distribution

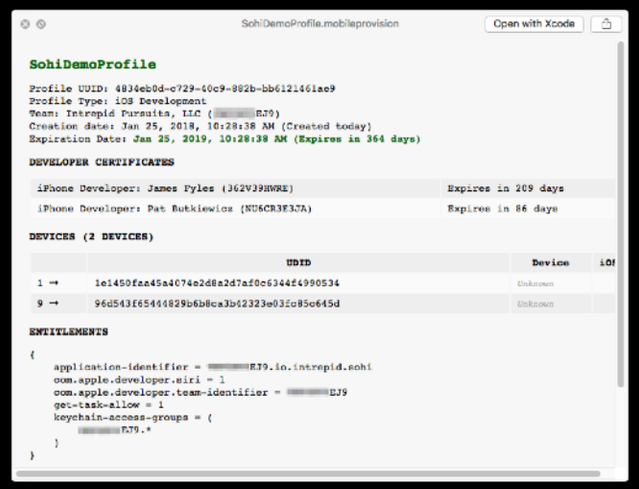
PROVISIONING PROFILES

- Team ID
- Bundle ID
- App ID
- Device ID(s)
- Entitlements

- **Team ID:** Unique ID for each development team.
- **Bundle ID:** Used so the app can be identified without ambiguity. Usually a reverse domain identifier such as (com.apple.appName)
- **App ID:** Combination of the team ID and bundle ID
  - Teams may develop more than one app.
  - Team ID is supplied by Apple.
- **Device IDs:** Every iOS device has a unique ID (UDID). Used to identify which devices are able to use this profile
- **Entitlements:** Manages access to system resources.

PROVISIONING PROFILES

```
$ security cms -D -i example.mobileprovision
```



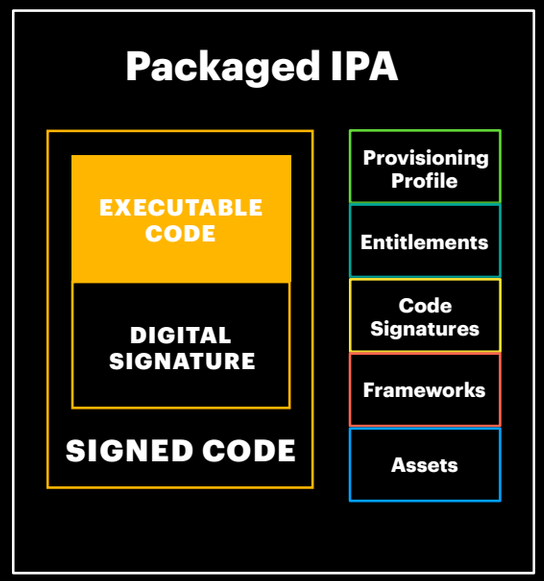
- Not a plist — Comes in a CMS format
  - Cryptographic Message Syntax (RFC 3852)
  - CMS allows the file to be signed so that it cannot be changed after it's generated in the developer portal
  - The "DeveloperCertificates" key lists all certificates allowed to sign the app.
    - **IF YOU SIGN THE APP WITH A CERTIFICATE NOT IN THIS LIST, IT WILL NOT RUN, PERIOD.**
  - The "ProvisionedDevices" key lists all the devices you set up with the provisioning profile.
  - The profile is packaged with the IPA and installed onto the device or uploaded to apple.

OTHER RESOURCES

- iOS apps are more than executable code. XIBs, storyboards, images, translation files, archives, are bundled with the app.
- Additional files and assets signed
- Signatures stored in “_CodeSignatures/” directory inside IPA

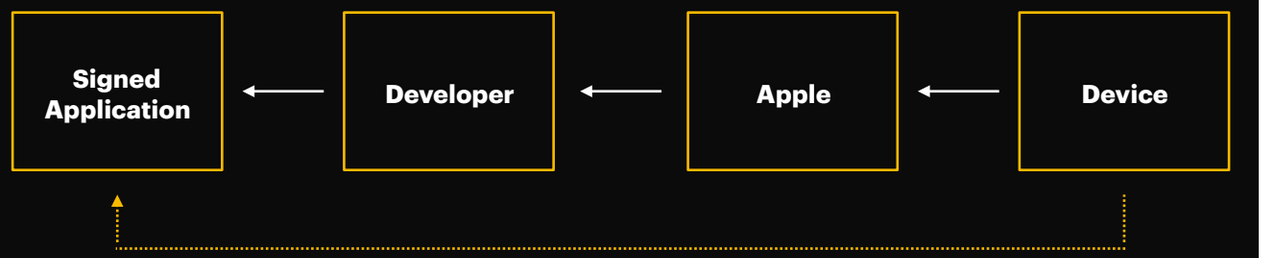
- Used to store the signature of all the files in the bundle that are signed

A COMPLETE IPA



- Everything we need to trust this application to run
  - Apple trusts and knows the person who signed the code because they've created a certificate with apple, which apple has signed.
  - The provisioning profile, ALSO signed by apple, will let us know if the digital signature, and the entitlements are valid for the App ID.

CHAIN OF TRUST



1. The trust chain shows why devices can trust signed applications

04 COMMON ISSUES

Q: I HAVE DOWNLOADED THE PROVISIONING PROFILE AND CERTIFICATES FROM THE DEVELOPER PORTAL, BUT I STILL CAN'T SIGN MY APP. WHY?

A: You don't have the private key! You can get the private key from. The person who originally generated the certificate, or you can revoke the certificate and revoke a new one.

Q: MY IOS DISTRIBUTION CERTIFICATE EXPIRED, WILL MY APPS STILL WORK?

A: As long as you stay enrolled in the developer program, Apple will keep your apps on the app store. Apple resigns your apps with their own certificate when you submit them to the app store, so an expired certificate on your end won’t affect the end users.

Q: I UPDATED THE ENTITLEMENTS ON MY APP ID, BUT BUILDING MY APP ON XCODE STILL DOESN'T ALLOW ME. TO USE THAT ENTITLEMENT.

A: Entitlements are written to the provisioning profile, which is signed by Apple. Because it's signed, every time we change something associated with the provisioning profile, we must regenerate it, download it, and import it into Xcode.

05 CLOSING THOUGHTS

Code Signing can be a pain, but the extra effort is worth it for the security of the device, and your users.

Thank You!